UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

PROGRAMAÇÃO HOMOGÉNEA DE REDES DE SENSORES USANDO O *MIDDLEWARE* MUFFIN

Rui José Laranjeira Pires

PROJETO
MESTRADO EM INFORMÁTICA

2013

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

PROGRAMAÇÃO HOMOGÉNEA DE REDES DE SENSORES USANDO O *MIDDLEWARE* MUFFIN

Rui José Laranjeira Pires

PROJETO

Projecto orientado pela Profª. Doutora Maria Dulce Pedroso Domingos
e pelo Prof. Doutor Francisco Cipriano da Cunha Martins

MESTRADO EM INFORMÁTICA

2013

Agradecimentos

Começo por agradecer aos meus orientadores Professora Doutora Dulce Domingos e Professor Doutor Francisco Martins pela oportunidade de participar neste projeto. Agradeço a sua orientação no decorrer dos trabalhos e também a sua constante disponibilidade, que em muito contribuiu para o sucesso deste ano que passou. Agradeço em especial ao Professor Francisco, não só por este ano de trabalho, mas também por todos os conselhos e conhecimentos transmitidos ao longo do meu percurso académico, que em muito contribuíram para as minhas decisões.

Agradeço também ao Bruno Valente pela sua contribuição com o MuFFIN e agradeço o apoio dado pela FCT através do projeto PATI (PTDC/EIAEIA/103751/2008) e do projecto MACAW (PTDC/EIA-EIA/115730/2009) e também pelos programas multianual LaSIGE (U I 408 - 2011-2013, refª PEst-OE/EEI/UI0408/2011).

Quero também agradecer a todos os meus amigos que em muito têm contribuído e ajudado a transformar-me na pessoa que sou hoje em dia. De todos eles agradeço em especial aos membros da "Ordem da Piñacolada" pelos bons momentos destes últimos 5 anos e também aos "Soldadores" que é como se fosse uma família. Um agradecimento muito especial aos meus amigos Carlos Mão de Ferro e Vitor Oliveira pela parceria estabelecida em muitos dos projetos realizados, a qual proporcionou muitas noitadas de esforço e dedicação, mas acima de tudo uma grande amizade. Agradeço também ao Miguel Garcia pelas "doses de cafeína" deste último ano e é claro pelas suas discussões, conselhos e experiência que de alguma forma sempre ajudaram.

Por último lugar, mas não o menos importante, quero agradecer que me têm apoiado em todos os momentos da minha vida, a minha família! Um muito obrigado aos meus pais por todos os sacrifícios a que se sujeitaram para me poderem proporcionar uma educação e qualidade de vida que infelizmente não puderam ter. Um grande obrigado ao meu irmão por todos estes anos de cumplicidade, partilha e grande amizade, os quais resultaram de momentos inesquecíveis. Deixo também uma agradecimento muito especial à minha avó Felismina por todos os seus conselhos, ensinamentos e exemplo de vida. Ainda na família agradeço a todos os meus tios e é claro à "primalhada", pela família unida que somos e que em muito me tem apoiado na maior parte dos momentos da minha vida.

A todos um MUITO OBRIGADO!

Para os meus pais e irmão.

Resumo

No contexto das redes de sensores sem fios, que atualmente são um tópico de bastante destaque e alvo de investigação em diversos domínios, surge a necessidade de programação dos seus dispositivos.

Os serviços *web* são utilizados para disponibilizar uma interface o mais homogénea possível das funcionalidades das redes de sensores. O *middleware* MuFFIN permite inclusive a (re)programação remota do comportamento de redes de sensores através de serviços *web*. No entanto, esta reprogramação está dependente das características do *hardware* ou das linguagens de programação disponibilizadas pelos fabricantes.

De modo a generalizar esta funcionalidade, propomos uma extensão ao *middleware* para incluir a execução de código em substituição dos sensores, quando estes não são reprogramáveis. Como prova de conceito, neste projeto utilizamos o *middleware* MuFFIN e a linguagem de programação de sensores Callas conjuntamente com a sua máquina virtual. Adicionalmente estendemos o MuFFIN com um componente que permite a comunicação entre duas redes de sensores sem que as mensagens trocadas extravasem o *middleware*, com o objetivo de criar uma forma de comunicação entre redes de sensores e atuadores, sem que esta tenha necessidade de passar pelas aplicações cliente.

Palavras-chave: *Middleware*, Máquinas virtuais, Reprogramação, Redes de sensores, Callas

Abstract

In the context of wireless sensor networks, which are currently a topic of research in various fields, there is the need to program their devices.

Web services are used to provide an homogeneous interface to sensor networks. The MuFFIN middleware even supports the remote (re)programming of sensors via web services. However, this (re)programming functionality dependents on the hardware characteristics as well as on the programming languages manufacturers provide.

In order to generalize this functionality, we propose a middleware extension which executes the code on behalf of sensor devices, in case they are not (re)programmable. As a proof of concept we use the MuFFIN middleware and the sensor programming language Callas together with its virtual machine.

Additionally, we extend the MuFFIN with a new component that supports the communication between two sensor networks. This way, messages can flow from one network to another one without the intervention of the client application, reducing the number of messages exchanged between sensor networks and client application.

Keywords: Middleware, Virtual Machines, Re-programming, Sensor Networks, Callas

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste capítulo são apresentados a motivação e os objetivos deste projeto. Apresentamos também as contribuições que resultaram do nosso trabalho e a estrutura dos restantes capítulos deste documento.

1.1 Motivação

Com os avanços tecnológicos nas áreas da eletrónica e da comunicação sem fios, surge o conceito da Internet das Coisas, que é um paradigma que tem como objetivo disponibilizar os elementos observados no mundo real e as suas representações no mundo digital.

Existem várias formas de recolher o estado dos elementos que são observados, passando por sistemas de códigos de barras, sistemas de *Radio Frequency Identification* (RFID), *Near Field Communication* (NFC) ou até sistemas com maior poder computacional que permitem a integração de componentes, como sistemas de navegação e diversos sensores de monitorização [6]. No campo destes dispositivos de monitorização, as redes de sensores têm especial interesse devido às suas capacidades de processamento e de troca de informação.

Uma rede de sensores é formada por um conjunto de dispositivos interligados entre si, tendo como objetivo efetuar a recolha de informações (por exemplo temperatura ou humidade) do meio em que estão instaladas. Dadas as suas capacidades, estes dispositivos são bastante utilizados em diversas áreas como segurança, defesa, gestão de tráfego, automatização de processos, entre outros. Na constituição destas redes existem dispositivos com capacidades distintas. Por exemplo, alguns que enviam periodicamente valores, outros que têm a capacidade de responder a pedidos e ainda outros que, por possuírem uma maior capacidade de processamento, podem ser utilizados para computações mais complexas. Um possível modo de interligar estes dispositivos é através da utilização de uma rede sem fios (*WSN - Wireless Sensor Network*), como por exemplo por meio de ZigBee [2].

Tendo em conta a grande variedade de dispositivos que existe no mercado (e.g., Ar-

duino [3], Crossbow [11]), os serviços *web* têm sido utilizados de forma a disponibilizar uma interface homogénea para acesso à informação e às funcionalidades dos sensores. Estes serviços *web* são disponibilizados nos próprios sensores ou em *middleware* [38]. O MuFFIN[34][35] é um exemplo deste tipo de *middleware*, o qual disponibiliza ainda um serviço *web* para reprogramação remota de sensores, no caso destes suportarem esta funcionalidade. A disponibilização deste serviço está dependente das capacidades dos sensores ao nível do *hardware* e do *software*.

De forma a generalizar a funcionalidade de reprogramação para todos os sensores, propomos estender o *middleware* de modo a incluir a capacidade de executar código em vez dos sensores que não são reprogramáveis. Neste projeto estendemos o *middleware* MuFFIN com a máquina virtual do Callas, como prova de conceito. Esta integração inclui a resolução dos desafios inerentes à execução de código dos sensores pelo *middleware*, nomeadamente as chamadas ao sistema operativo.

Adicionalmente, para permitir a comunicação entre redes de sensores, o *middleware* terá um novo componente para encaminhar mensagens enviadas de uma rede origem para outra de destino. Desta forma é possível reduzir o número de mensagens para as aplicações de alto nível, descentralizando parte do processo de decisão para o *middleware* ou nas redes de sensores [20].

1.2 Objetivos

Os principais objetivos deste trabalho são criar um sistema que permita realizar a programação de dispositivos com apenas uma linguagem e um outro sistema que permite a comunicação entre redes de sensores ao nível do *middleware*.

Seguidamente são detalhados estes objetivos assim como os respetivos desafios e a abordagem seguida para cada um deles.

1. **Analisar as tecnologias utilizadas pelo MuFFUN:** realização de um estudo sobre as tecnologias utilizadas pelo *middleware* e respectivas versões.
   Desafios:

   - identificação das novas versões das tecnologias.
   - identificação de possíveis problemas nas novas versões.
   - atualização do MuFFIN de modo a incluir as versões mais recentes das tecnologias.

   Abordagem: realizar um estudo sobre as tecnologias usadas pelo MuFFIN, de modo a averiguar as versões das tecnologias utilizadas. Proceder à migração do MuFFIN para estas novas versões.

2. **Estender o *middleware* de modo a suportar a reprogramação de sensores independentemente das suas capacidades:** permite que o *middleware* execute código dos sensores em sua substituição. No protótipo desenvolvido, o MuFFIN foi estendido de modo a executar código escrito na linguagem Callas.
   Desafios:

   - desenvolvimento de um componente para embeber a máquina virtual do Callas.
   - integração do componente desenvolvido com os restantes componentes do MuFFIN.
   - atualização dos serviços existentes para interação com a máquina virtual.

   Abordagem: desenvolver um novo componente para embeber a máquina virtual de Callas. Este componente segue a arquitetura *Open Services Gateway initiative* (OSGi) [1] utilizada pelos componentes do MuFFIN, permitindo uma melhor integração com o sistema.

3. **Identificar os intervenientes e as propriedades das redes de sensores:** reconhecer a estrutura da rede e quais os dispositivos que a constituem.
   Desafio:

   - utilização de uma norma que permite realizar a descrição das redes de dispositivos ao nível do *middleware*.

   Abordagem: criar um mecanismo capaz de interpretar a norma SensorML [29], permitindo ler o conteúdo dos ficheiros que descrevem a rede. Também é necessário alterar o serviço *web* de instalação de pontos de acesso (*gateway*) para cada rede, o qual inclui a informação SensorML que descreve a rede com a qual o *gateway* vai interagir.

4. **Sistema de comunicação entre redes de sensores:** permitir estabelecer a comunicação entre redes independentes ao nível do *middleware*.
   Desafios:

   - criação de um sistema para estabelecer a comunicação entre duas redes.
   - inclusão de um mecanismo para realizar a adaptação de protocolos utilizados pelas redes.
   - criação de uma forma de encaminhamento de mensagens de uma rede de origem para uma rede de destino.

   Abordagem: desenvolver um mecanismo de comunicação que segue o paradigma de publicação/subscrição, onde é possível estabelecer comunicação ponto a ponto

entre duas redes. Criar um serviço *web* que permite a instalação de conversores por parte dos clientes do *middleware*. Cada um destes elementos tem como função realizar a adaptação das mensagens enviadas de uma rede para outra.

1.3 Contribuições

Do trabalho realizado, resultam as seguintes contribuições:

Atualização das tecnologias utilizadas pelo MuFFIN: proporcionou melhoramentos em alguns aspetos de segurança das tecnologias utilizadas, como é o caso dos mecanismos de autenticação do ActiveMQ e resolução de alguns erros existentes, como por exemplo um problema associado à ferramenta XML Beans, que anteriormente foi resolvido através da substituição de ficheiros no núcleo do Fuse ESB, prática altamente desaconselhada.

Reprogramação de dispositivos com apenas uma linguagem: possibilita a reprogramação remota de redes de sensores com recurso a apenas uma linguagem, independentemente das suas capacidades. A solução inclui o motor de execução de uma linguagem ao nível do *middleware* para que este possa executar o código destinado aos sensores.

Reprogramação de dispositivos com a linguagem Callas: como prova de conceito da contribuição anterior, foi realizada uma extensão ao *middleware* MuFFIN para incluir a máquina virtual de Callas de forma a possibilitar a reprogramação remota de redes de sensores com recurso a apenas esta linguagem. Assim, se os sensores suportarem esta linguagem, o código é instalado diretamente nestes; caso contrário, o código dos sensores é executado ao nível do *middleware*.

Comunicação de redes de dispositivos ao nível do *middleware*: permite a comunicação entre duas ou mais redes ao nível do *middleware*, descentralizando parte do processo de decisão para o *middleware* ou para as redes de sensores, em vez de ficar centralizado nas aplicações cliente.

1.4 Estrutura do documento

Este documento encontra-se organizado da seguinte forma: no Capítulo 2 são descritos os conceitos das tecnologias utilizadas pelo MuFFIN, bem como o funcionamento e arquitetura deste *middleware*. Neste capítulo é também descrito o trabalho relacionado, nomeadamente sobre máquinas virtuais, *middleware* e simulação de sensores. O Capítulo 3 apresenta a forma de funcionamento do novo componente do MuFFIN, que permite a

execução de código ao nível do *middleware*. É também realizada uma descrição da arquitetura deste componente e dos seus detalhes de implementação. No Capítulo 4 descrevemos o novo componente que permite estabelecer a comunicação entre as redes de sensores. Tal como no Capítulo 3, no Capítulo 4 são apresentados os detalhes da arquitetura do componente, bem como os seus detalhes de implementação. No Capítulo 5 são apresentados e discutidos os valores obtidos nos testes que foram realizados aos novos componentes. Também é descrita uma experiência para ilustrar um caso real de utilização do MuFFIN, que incide principalmente nas novas funcionalidades, de forma a permitir uma melhor visão do seu funcionamento. Por fim, no Capítulo 6 são apresentadas as conclusões deste projeto e o respetivo trabalho futuro que poderá ser desenvolvido.

Capítulo 2

Trabalho relacionado

Este capítulo descreve os sistemas de *middleware* relacionados com o projeto, em particular o MuFFIN [34][35], que é o *middleware* utilizado neste trabalho como prova de conceito das nossas propostas. Apresenta também projetos relacionados com a simulação de sensores e sobre máquinas virtuais, para encontrar aspetos em comum e realizar um enquadramento com o problema apresentado. Por fim, é abordado o tema da máquina virtual de Callas [27], elemento agora integrado no MuFFIN.

2.1 *Middleware*

Esta secção descreve sistemas de *middleware* que possibilitam a interação com redes de sensores.

- Choon-Sung Nam *et al.* [9] propõe um *middleware* com uma arquitetura orientada a eventos (EDA, do inglês Event Driven Architecture) que utiliza o paradigma de publicação/subscrição, possibilitando às aplicações cliente a subscrição de tópicos do seu interesse e que recebam notificações quando é publicada informação com esses tópicos. Esta comunicação permite reduzir a quantidade de mensagens trocadas, face aos sistemas de espera activa usados em cenários onde a comunicação é síncrona. A arquitectura apresentada está limitada a um conjunto estático de tópicos, fazendo com que não haja uma gestão modular do sistema, nem o acesso remoto através de serviços. O MuFFIN implementa uma arquitectura semelhante à apresentada, no entanto foram eliminadas as limitações relacionadas com a falta de modularidade por meio de uma rede dinâmica de filtros que é construída ao longo do tempo.

- Hervé Paulino e João Santos [31] criaram um *middleware* que permite a interação entre redes de sensores e aplicações cliente através de serviços *web*. Para que os utilizadores possam operar sobre os dados recebidos das redes, é apresentada a sintaxe de uma linguagem proposta que permite criar filtros sobre os dados recebidos.

Nesta abordagem foram tidos em conta os padrões *Sensor Web Enablement* (SWE) [30], embora estes não tenham sido implementados, pois foram utilizadas soluções que funcionam de forma semelhante. Para além desta limitação, não é possível a este sistema realizar a composição de filtros de dados.

- O SenseWeb [23] é um *middleware* que tem como objetivo disponibilizar serviços *web* que permitem a várias aplicações obter informação de diferentes dispositivos de uma forma uniforme. Esta solução permite a interação com dispositivos fixos e móveis por intermédio de uma interface comum a todos os dispositivos (*gateway*). Os dados recebidos das redes de dispositivos são armazenados numa base de dados. Esta abordagem tem a limitação de não serem usadas normas no armazenamento e disponibilização de informação, e também o facto de não permitir a reprogramação de dispositivos.

- O *Middleware Framework For the Internet of thiNgs* (MuFFIN) [34] [35] foi a plataforma escolhida para a implementação das funcionalidades de execução de sensores e comunicação entre redes de sensores ao nível do *middleware*, o qual se encontra descrito na secção 2.2. Esta escolha deveu-se a este *middleware* ser uma plataforma que segue as normas SWE e por ser capaz de a interagir e reprogramar redes de sensores através da utilização de serviços *web*.

2.2 MuFFIN

Nesta secção são apresentadas as tecnologias utilizadas na construção do MuFFIN, e também são descritas as suas funcionalidades e arquitetura.

2.2.1 O MuFFIN

O MuFFIN é um *middleware* que permite que aplicações de alto nível, comuniquem com redes de sensores utilizando serviços *web*. O *middleware* segue as normas SWE da *Open Geospatial Consortium* OGC, nomeadamente o *Sensor Observasion Service* (SOS), no que diz respeito à disponibilização de informação para o utilizador, *Observation and Mesurements* (O&M) para armazenamento dos dados vindos das redes [30], e o *Web Service Notification* (WS-N) [22], para notificação dos clientes *web*

Um dos principais objetivos deste *middleware* é permitir a reprogramação de dispositivos em tempo de execução. O sistema segue o paradigma Arquitectura para Dispositivos Orientados-a-Serviços (SODA, do inglês *Service Oriented Device Architecture*) [12], fornecendo as funcionalidades dos dispositivos por meio de serviços *web*. Deste modo, é disponibilizada uma interface bem definida de acesso ao *hardware*, independente da linguagem de programação e da plataforma em que executam.

Como muitos dispositivos não permitem ser reprogramados remotamente ou em tempo de execução, o *middleware* possui uma funcionalidade que permite minimizar essa limitação. O MuFFIN oferece um serviço que permite instalar código na rede de sensores caso os dispositivos o suportem. Adicionalmente é possível instalar módulos ao nível do *middleware* que operam como filtros ou agregadores de informação enviada pelas redes de sensores. Ainda assim, estas características não tornam a utilização do MuFFIN muito uniforme, pois o utilizador necessita saber se a rede é reprogramável e quais as suas características para enviar o código adequado à rede. No caso do utilizador utilizar módulos, este tem de utilizar a linguagem *Java* para desenvolver os módulos a serem instalados no *middleware*.

O MuFFIN é uma aplicação modular baseada em arquitectura de software orientada-a-serviços (SOA, do inglês *Service-Oriented Architecture*)[36], constituída por sete componentes que são apresentados na secção seguinte. A gestão e desacoplamento dos módulos é garantida e facilitada graças à arquitectura OSGi disponibilizada pelo Fuse ESB [33], permitindo que sejam adicionados ou alterados módulos no *middleware* em tempo de execução. Cada um dos módulos tem disponível uma interface de comunicação que os restantes podem utilizar.

2.2.2 Arquitetura

A Figura 2.1, apresenta a arquitetura do MuFFIN e a forma como os seus componentes comunicam.

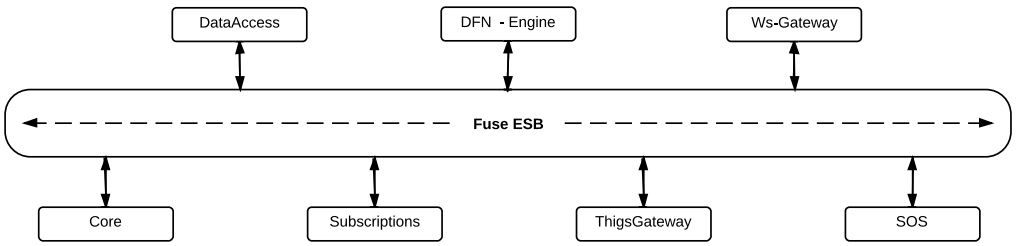

Figura 2.1: Arquitetura do MuFFIN

Descrição dos componentes do MuFFIN:

- **WS-Gateway**: apresenta as funcionalidades do *middleware* sobre a forma de serviços *web*.

- **Core**: organiza os módulos utilizados na implementação dos serviços *web*. Este módulo depende diretamente dos módulos *Subscriptions* e *DFN-Engine*, pois é para estes que o Core envia a informação dos pedidos feitos pelas aplicações cliente.

- **ThingsGateway**: permite carregar pontos de acesso a redes de sensores de forma dinâmica. Este componente interage com a camada WS-Gateway para receber os pedidos de instalação de *gateways* e de código a ser instalado nas redes de sensores.

- **DFN-Engine**: gere a rede de fluxo de dados que é utilizada para filtrar e agregar os dados recebidos da rede de sensores. O *DFN-Engine* é também responsável por receber, instanciar e gerir os módulos enviados pelas aplicações de alto nível.

- **SOS**: Este elemento é responsável por realizar o processamento de documentos XML da norma SOS, como por exemplo documentos de inserção de observações e leitura de observações. Este componente possui os interpretadores XML necessários à implementação da especificação SOS.

- **Subscriptions**: gere as subscrições dos serviços *web* do MuFFIN realizadas pelos clientes, ou seja, guarda os dados das publicações para que seja possível notificar os subscritores desses dados.

- **DataAccess**: disponibiliza os elementos de acesso à camada de persistência do MuFFIN.

2.2.3 Tecnologias utilizadas

A construção do MuFFIN está assente num conjunto de mecanismos que disponibilizam várias funcionalidades, como por exemplo, o acesso à base de dados ou a criação de documentos XML. Seguidamente são descritas as tecnologias com as quais foi necessário interagir durante o desenvolvimentos dos novos componentes.

- **Apache Maven [15]:** Facilita a gestão de projetos e a agilização dos processos de desenvolvimento.

- **Fuse ESB (Entreprise Service Bus) [33]:** É a plataforma de integração onde foi feita a instalação do MuFFIN. Esta ferramenta é baseada no *Apache ServiceMix* [16], implementando as funcionalidades da arquitetura OSGi [1]. O Fuse ESB permite a integração de várias ferramentas, como por exemplo o Maven que permite a instalação e gestão de módulos a partir da consola do Fuse ESB.

- **Apache ActiveMQ [13]:** Esta tecnologia permite a comunicação entre componentes de *software*, graças à implementação do JMS (*Java Message Service*). Dessa forma, o ActiveMQ torna-se um intermediário de comunicação entre componentes. O MuFFIN utiliza este *software* na gestão das filas de mensagens e dos tópicos de interesse.

- **Apache XML Beans [17]:** É uma ferramenta que permite transformar esquemas XML em classes Java e realizar a produção de documentos XML a partir de uma estrutura de objetos Java que representam o esquema XML correspondente. Dessa forma, permite tratar o XML enviado pelos clientes e também criar as respetivas respostas neste formato.

- **Apache CXF [14]:** Permite-nos realizar a construção de serviços *web* através da utilização de anotações nas classes que são apresentadas como sendo serviços.

- **JBoss Hibernate [10]:** Permite a criação das entidades responsáveis pelo acesso à camada de persistência do MuFFIN. Esta componente implementa o *Java Persistence API* (JPA), permitindo representar as entidades e as relações existentes por meio de classes Java, tornando assim o acesso à camada de persistência independente do tipo de Sistema de Gestão de Base de Dados (SGBD) utilizado.

- **MySQL [28]:** É o SGBD utilizado pelo MuFFIN na sua camada de persistência. Como visto no ponto anterior este poderia ser um outro SGBG, pois, graças ao *Hibernate* existe uma transparência entre a camada de persistência e a camada de negócio.

- **SensorML [29]:** A *Sensor Model Language* é uma linguagem para a modelação de sistemas de sensores, que especifica modelos e esquemas em XML, com os quais é possível realizar a descrição de redes de sensores e as capacidades dos elementos da rede. O objetivo desta linguagem é uniformizar a especificação e classificação dos dispositivos, tornando genérica a identificação e utilização de cada tipo de dispositivo.

2.3 Simulação de redes de sensores

Em [37] é apresentada uma abordagem de integração de redes de sensores simuladas com fluxos de trabalho. Para efeitos de simulação é utilizado o Visual Sense [7], sendo possível realizar execuções do sistema em que são utilizados dados simulados, facilitando as experiências com novos cenários.

Neste trabalho é também utilizada a linguagem Callas para a simulação de sensores, o que nos permite identificar problemas genéricos de simulação de redes de sensores, assim como de simulações de aplicações codificadas em Callas. Neste trabalho foi realizada uma integração da máquina virtual com o *Visual Sense* surgindo a necessidade de realizar alterações à máquina virtual de Callas. Este trabalho permitiu conhecer melhor a arquitetura da máquina virtual, ajudando a perceber como realizar a sua integração no MuFFIN.

2.4 Máquinas virtuais

Uma máquina virtual é um componente capaz de correr aplicações de forma mediada. Para conseguir esta mediação, uma máquina virtual utiliza os recursos existentes na máquina real utilizando técnicas que permitem correr as aplicações de uma forma isolada, sem interferir na execução do que se encontra na máquina real.

As máquinas virtuais podem ser classificadas em dois tipos: máquina virtuais de tipo 1 que são as instaladas e executadas diretamente sobre o hardware; e as de tipo 2 que são instaladas e executadas sobre um sistema operativo, denominado de anfitrião (*host*) [26].

Seguidamente serão apresentados alguns exemplos de máquinas virtuais existentes, que se enquadram no conceito deste projeto.

- **TakaTuka [4]:** É uma *Java Virtual Machine* (JVM) capaz de executar em pequenos sistemas embebidos e micro controladores com uma capacidade de memória de apenas 4kb. Esta máquina virtual tem a vantagem de tornar possível a execução de código *Java* em dispositivos com recursos bastante limitados.

- **PyMite [19]:** Máquina virtual de Python capaz de executar uma parte significativa das instruções dessa linguagem em dispositivos sem sistema operativo e bastante limitados a nível de recursos.

- **Nano VM [21]:** É uma JVM para dispositivos que possuam processadores da família AVR CPU [5]. Esta máquina virtual substitui o *firmware* original dos dispositivos, sendo executada diretamente sobre o hardware dos dispositivos. Devido às limitações dos processadores, esta máquina virtual apenas é capaz de correr um pequeno conjunto das instruções da JVM original.

- **Squawk [32]:** É uma JVM concebida para executar sem sistema operativo, tendo sido testada nos dispositivos Sun SPOT[24]. Esta máquina virtual tem ainda capacidade de correr várias aplicações em simultâneo.

- **Maté [25]** É um sistema de comunicação assente em uma máquina virtual destinado a redes de sensores, permitindo tornar programas complexos em programas pequenos com o objetivo de reduzir os custos energéticos associados à transmissão dos programas. Esta máquina virtual é executada em TinyOS necessitando apenas de 1kb de memória RAM e 16kb de memória para instruções.

- **Darjeeling [8]:** É uma máquina virtual baseada na JVM, capaz de excutar parte das instruções da linguagem Java. De forma a ser mais eficiente foi concebida para correr em micro-controladores com arquiteturas de 8 e 16 bits e com 2 a 10kb de memória RAM.

Todas as máquinas virtuais anteriormente descritas são destinadas a dispositivos de pequenas dimensões, no entanto estas possuem algumas limitações. Umas porque estão dependentes da plataforma, como é o caso da Nano VM e Squawk, outras porque executam parte das instruções da linguagem utilizada, como a PyMite e Darjeeling, ou porque estão dependentes de um sistema operativo que é o caso da Maté.

Na secção 2.5 é apresentada a linguagem Callas, a sua máquina virtual e as razões que nos levaram a optar pela sua escolha de utilização neste projeto.

2.5 Callas

Nesta secção são apresentadas as características da linguagem Callas e é feita uma descrição da arquitetura da máquina virtual de Callas.

2.5.1 A linguagem

A linguagem de programação de sensores Callas [27] tem como objetivo estabelecer uma base para o desenvolvimento de linguagens de programação e sistemas em tempo de execução para redes WSN. Esta linguagem pode ser utilizada diretamente como linguagem de programação de uma rede de sensores, ou servir de intermediário com linguagens de mais alto nível.

É uma linguagem tipificada, permitindo que os programas sejam construídos segundo tipificação correta, não causando erros em tempo de execução. Outra característica desta linguagem, é a possibilidade de reprogramação remota de sensores, permitindo corrigir problemas ou realizar atualizações ao sistema, sem a necessidade de aceder fisicamente aos dispositivos.

As redes de sensores assentes na linguagem Callas têm como particularidade o facto de todos os dispositivos implementarem a mesma interface, embora possam assumir comportamentos diferentes dependendo da implementação das interfaces presentes nos dispositivos.

As aplicações desenvolvidas em Callas executam na máquina virtual de Callas (CVM), descrita na Secção 2.5.2, a qual permite realizar uma abstração relativamente ao *hardware* dos dispositivos onde a máquina virtual está instalada. Uma vez que não é o intuito deste projeto desenvolver aplicações em Callas não iremos abordar questões que dizem respeito à linguagem.

2.5.2 A máquina virtual

A arquitetura da CVM [37] é constituída por três *threads*, como pode ser observado na Figura 2.2. Uma *thread* que funciona como interpretador, outra responsável por receber mensagens (código binário) e uma responsável por enviar mensagens. A *thread* que

funciona como interpretador pode ser vista como a *thread* principal, sendo constituída por uma fila de entrada, outra de saída e um interpretador. Tem como principal objetivo avaliar os programas/mensagens que estão na fila de entrada e após o processamento, coloca as respostas na fila de saída. As outras duas *threads*, servem de intermediário entre a máquina virtual e os dispositivos, sendo responsáveis por adaptar as mensagens de acordo com as mensagens de saída ou entrada. A linguagem Callas utiliza também uma instrução (*extern*) que permite realizar chamadas ao sistema dos dispositivos.

A escolha da linguagem Callas e respetiva máquina virtual deveu-se ao facto desta linguagem ter sido desenvolvida com o objetivo de ser utilizada na programação de redes de sensores, e a sua máquina virtual tem a vantagem de não depender de nenhuma plataforma ou sistema operativo, possibilitando uma melhor integração com os sistemas onde esta seja aplicada.

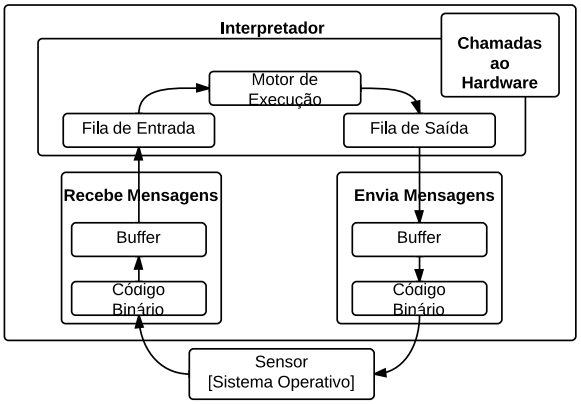

Figura 2.2: Arquitetura da máquina virtual de Callas

Neste capítulo foram apresentados alguns sistemas de *middleware* relacionados com o projeto. Foram também apresentados projetos relacionados com a simulação de sensores e sobre máquinas virtuais, permitindo realizar um enquadramento com o problema apresentado. O capítulo apresenta ainda uma descrição mais detalhada do *middleware* MuFFIN, da linguagem Callas e da sua máquina virtual pois foram os elementos utilizados neste trabalho.

Nas secções seguintes iremos apresentar as novas funcionalidades de reprogramação de dispositivos com recurso a uma única linguagem e de comunicação de redes de sensores ao nível do *middleware*, sendo apresentado como prova de conceito o MuFFIN estendido com estas novas funcionalidades.

Capítulo 3

Execução virtual de redes de sensores

Neste capítulo é apresentado o conceito de execução virtual de redes de sensores ao nível do *middleware* de forma a permitir a reprogramação de todo o tipo de dispositivos com recurso a uma única linguagem. O capítulo apresenta também a concretização desta proposta como um novo componente do MuFFIN, que foi desenvolvido com o objetivo de permitir a reprogramação de todo o tipo de sensores.

3.1 Execução virtual de redes de sensores ao nível do *middleware*

A disponibilização para as aplicações da funcionalidade de (re)programação remota de sensores depende das linguagens de programação disponibilizadas pelos fabricantes, do *software* instalado nos sensores e das características do *hardware*. Consequentemente, nem todos os sensores são (re)programáveis remotamente e, considerando os sensores que apresentam esta funcionalidade, não existe uma linguagem comum para os (re)programar. De forma a disponibilizar a funcionalidade de (re)programação remota de sensores de forma homogénea às aplicações propomos a extensão do *middleware* de forma a executar o código em substituição dos sensores. Deste modo, o *middleware* terá de conhecer a configuração da rede de sensores e simular a execução do código assim como a interacção entre os sensores.

Na Figura 3.1 encontra-se ilustrada a ideia apresentada, em que existe uma rede não reprogramável que interage com o *middleware*. De forma a possibilitar a sua reprogramação, o comportamento desta rede é executado ao nível do *middleware* pelo componente de *Execução de sensores*, o qual recorre à base de dados do *middleware* para obter valores das observações dos sensores. Associados a esta ideia surgem os seguintes problemas:

1. **Problema 1 (representação dos sensores):** criar um mecanismo que permita criar vários sensores virtuais, em que cada um mantém o seu estado. O *middleware* tem de conhecer as capacidades das redes de sensores e a sua configuração. Em particular, este tem de ter informação sobre os sensores de uma rede que têm capacidade

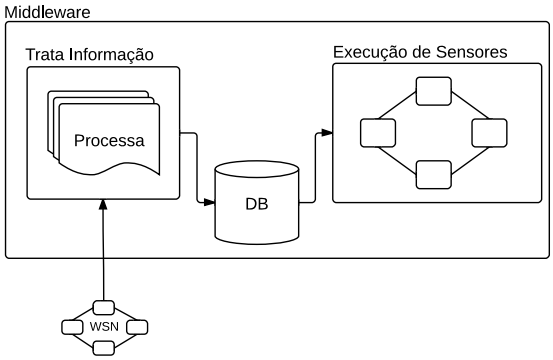

Figura 3.1: Esquema conceptual do sistema de execução de sensores

para executar código de modo a determinar se este será enviado para os sensores ou se será executado localmente, no *middleware*. A configuração da rede é necessária para se identificar o número de sensores a serem executados;

2. **Problema 2 (simular as leituras dos sensores):** dado que se tratam de sensores virtuais, há a necessidade de obter valores reais para representar as suas observações. Cada sensor tem de ser capaz de obter valores reais e atuais, de forma a que o processamento seja feito com dados válidos;

3. **Problema 3 (simular a comunicação entre sensores):** como se tratam de sensores virtuais é necessário simular a sua rede de comunicação, de maneira a que estes possam trocar informação.

3.2 MuFFIN-CVM

Como prova de conceito da ideia apresentada em 3.1, utilizamos o *middleware* MuFFIN e a linguagem de programação de sensores Callas juntamente com a sua máquina virtual.

O *middleware* MuFFIN já inclui um serviço web de (re)programação remota de sensores, estando a disponibilidade deste serviço dependente das características dos sensores. Em relação à opção pela linguagem Callas, esta justifica-se quer pelas suas características apresentadas na secção 2.5, quer pelo facto de podermos utilizar a respetiva máquina virtual para executar código Callas, bastando adaptá-la para que possa executar-se no contexto do MuFFIN.

A integração da Máquina Virtual do Callas no MuFFIN tem como objetivo a disponibilização do serviço *web* de (re)programação de sensores para todos os tipos de sensores numa linguagem comum, neste caso, a linguagem Callas. Desta forma o serviço *web* disponibilizado pelo MuFFIN torna transparente às aplicações o local onde é executado o código dos sensores.

Nas secções seguintes detalhamos a forma como foi efectuada a integração da máquina virtual do Callas no Muffin.

3.2.1 Integração da máquina virtual de Callas no MuFFIN

O MuFFIN permitia a reprogramação remota de sensores através de serviços *web*, para redes de sensores que suportem essa funcionalidade. De forma a estender esta funcionalidade a todas as redes, integramos a CVM no MuFFIN (componente MuFFIN-CVM). A Figura 3.2 apresenta a arquitetura deste novo componente.

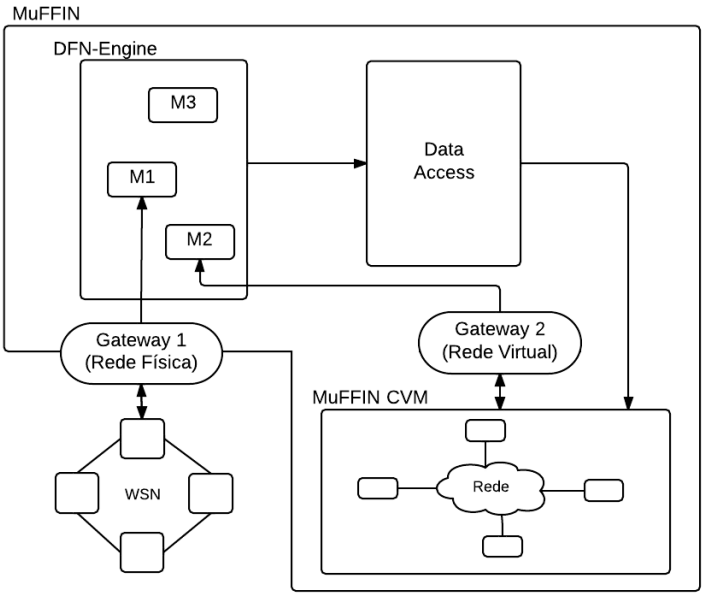

Figura 3.2: Integração do componente MuFFIN-CVM no *middleware*

A interacção com o MuFFIN-CVM é estabelecida por um *gateway*, através do qual são executados os pedidos de reprogramação tal como acontece no caso das redes físicas. Esta decisão foi tomada para garantir transparência para o utilizador, pois independentemente de ser uma rede real ou uma rede virtual, quando se pretende reprogramar uma rede, é utilizado o mesmo serviço *web*, sendo abstraído o tipo de rede para o utilizador. As Figuras 3.3 e 3.4 mostram o funcionamento do antigo e novo serviço de reprogramação, respetivamente. Através delas é possível observar que foi mantida a mesma interface. Apenas houve a necessidade de criar uma forma de identificar qual o tipo do *gateway*, embora isso seja transparente para o utilizador, pois foi concretizado no componente *Things Gateway*.

A Figura 3.2 ilustra a forma como o componente MuFFIN-CVM se encontra integrado no *middleware* e a respectiva arquitetura, obtida através da adaptação da máquina virtual original descrita em 2.5.2. Seguidamente apresentamos as decisões tomadas para resolver os problemas identificados na secção 3.1:

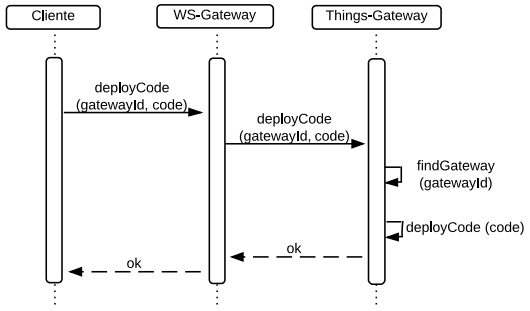

Figura 3.3: Antigo serviço *web* de reprogramação

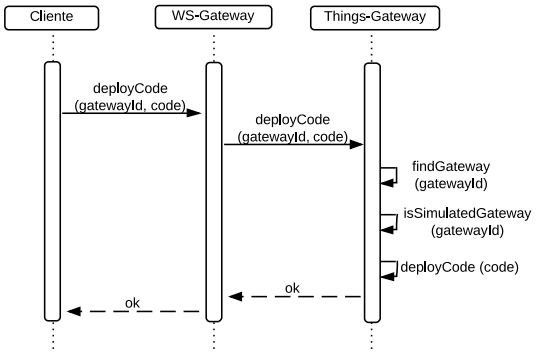

Figura 3.4: Novo serviço *web* de reprogramação

Representação dos sensores: Optou-se por concentrar o conhecimento sobre as capacidades e configuração das redes nos *gateways*, mantendo inalterados todos os restantes componentes do MuFFIN. De facto, são os *gateways* que encapsulam as especificidades dos sensores permitindo às restantes componentes do MuFFIN abstraírem-se dessas especificidades. Assim, quando é invocada a funcionalidade do *ThingsGateway* de enviar código para os sensores, é determinado se os sensores têm a capacidade de executar código Callas, caso isso seja possível o código é enviado para os sensores, caso contrário é enviado para a componente do MuFFIN que simula a sua execução (a MuFFIN-CVM). Neste caso é ainda necessário informar a MuFFIN-CVM sobre o número de sensores cuja execução deve simular, sendo utilizado um documento que segue a norma SensorML, o qual descreve os dispositivos constituintes da rede a ser simulada.

Simular as leituras dos sensores: na figura 3.2 é possível observar que o novo componente também interage com o componente responsável pela persistência dos dados. Esta interação foi criada para que cada sensor virtual possa aceder ao valores lidos pela rede real, pois como é executado código no lugar dos sensores, é necessário simular a interação com o sistema operativo ou *hardware*, como é o caso das observações realizadas pelos sensores. Desta forma, cada sensor virtual acede à base de dados do MuFFIN de modo a obter o valor da observação mais recente. Quando um sensor virtual realiza

um pedido à base de dados, este tem de indicar qual é o seu identificador único (por exemplo, o endereço MAC *(Media Access Control)*), de forma a que possa obter o valor da observação mais recente feita pelo sensor real. No decorrer da execução de cada sensor, sempre que cada um tem a necessidade de enviar informação destinada aos seus clientes, este envia-a para o *gateway*, sendo que a partir deste ponto a processamento da informação no MuFFIN continua como se de uma rede normal se tratasse.

Simular a comunicação entre sensores: para estabelecer a comunicação entre os sensores virtuais criamos um sistema onde todos os sensores virtuais escrevem e lêem mensagens. Este mecanismo é responsável por gerir as mensagens que circulam na ”rede”, garantindo que os sensores não leiam uma mensagem mais do que uma vez. No caso em que há um grande fluxo de mensagens, este mecanismo também é capaz de simular o caso em que são perdidas mensagens.

Utilizando a Figura 3.2 como exemplo, quando é invocado o serviço *web* de reprogramação e este pedido se destina a uma rede executada ao nível do *middleware*, o código é enviado para o componente MuFFIN-CVM através do *gateway 2*. Juntamente com o código enviado para a MuFFIN-CVM, é também enviada a informação sobre o número de sensores que fazem parte da rede, a qual foi obtida através dos dados segundo a norma SensorML, enviados aquando da instalação do *gateway*. Ao ser recebida esta informação na MuFFIN-CVM, são criados os sensores virtuais de acordo com os parâmetros enviados.

3.2.2 Implementação

Para a construção do componente MuFFIN-CVM, criamos um componente seguindo a arquitetura OSGi, para integrar com os componentes do MuFFIN que já existiam. Este novo componente possui uma interface, representada na Figura 3.5, que apenas disponibiliza um método, o qual permite realizar a execução de código.

```
public interface IMuffinCvm {

   public void runCode(String binaryPath, AbstractThing gateway, List<String> sensors);

}
```

Figura 3.5: Interface do componente MuFFIN-CVM

Na secção anterior vimos que é usado um serviço *web* para reprogramação de sensores independentemente das suas capacidades, de forma a tornar transparente a interação com as redes de sensores. Para tal foram necessárias algumas alterações, essencialmente no componente *Things Gateway*, pois é este o responsável pela interação com os *gateways*.

Desta forma, foi adicionado o mecanismo que permite verificar se um *gateway* se destina a interagir com uma rede real ou com uma virtual, quando se realiza um pedido de reprogramação. Com esta alteração é possível ao utilizador reprogramar qualquer tipo de rede com a mesma interface, não havendo a necessidade de identificar se está a lidar com uma rede física ou virtual. Este mecanismo de verificar o tipo de *gateway* é conseguido graças à informação que é passada quando se instala um *gateway*, pois é nesse momento que indicamos se o *gateway* vai comunicar com uma rede física ou virtual. Juntamente com essa informação seguem também os dados que descrevem a rede (seguindo a norma sensorML) com a qual o *gateway* vai interagir, para que se possa saber a estrutura de uma rede a ser criada (sensores virtuais), quando se trata de um *gateway* destinado a uma rede virtual.

A Figura 3.6 representa um exemplo de um documento no formato SensorML. Neste exemplo é possível observar algumas propriedades da rede descrita por este documento, nomeadamente o nome e quais os componentes constituintes do sistema descrito. Relativamente aos componentes é possível observar que para cada componente existe um identificador associado.

De forma a interpretar os dados dos documentos SensorML, foi criado um interpretador para este tipo de documentos. Este interpretador tem por base um conjunto de classes geradas pela ferramenta XML Beans. Para gerar estas classes, foram utilizados os ficheiros que possuem os esquemas da norma (ficheiros *XML Schema Definition, XSD*). Com este mecanismo, quando um documento é interpretado, é obtida a informação sobre os dispositivos que constituem a rede e quais os respetivos identificadores. Estes identificadores são utilizados nas leituras à base de dados do MuFFIN, fazendo com que cada execução de um sensor apenas obtenha valores da base de dados que dizem respeito ao seu identificador.

Quando é realizado um pedido de reprogramação e este se destina a um *gateway* que comunica com uma rede simulada, o componente *Things Gateway* escreve um ficheiro em disco com o código Callas destinado à reprogramação dos dispositivos. Esta operação é realizada para que no caso em que o MuFFIN por alguma razão precise de reiniciar os serviços, possa usar o código que estava em execução no momento em que foi reiniciado. Após ter sido escrito o ficheiro, é invocado o método presente na interface do MuFFIN-CVM (Figura 3.5), onde é enviado o caminho onde está o código Callas, o *gateway* que realiza a interação com a rede e a informação sobre a estrutura da rede virtual a criar.

De seguida apresentamos a forma como tratamos os problemas apresentados na secção 3.1:

Representação dos sensores: foi necessário alterar o núcleo da máquina virtual para criar uma forma de manter várias execuções de código em simultâneo, ou seja, manter o estado de cada um dos sensores virtuais. Inicialmente para representar cada um dos

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:SensorML xmlns:sml="http://www.opengis.net/sensorML/1.0.1"(...) >
   <sml:member>
      <sml:System gml:id="systemName">
         <sml:identification>
         </sml:identification>
                 <sml:components>
                    <sml:ComponentList>
                           <sml:component gml:id="B3310001">

                               (...)

                           </sml:component>
                           <sml:component gml:id="B3310002">

                               (...)

                           </sml:component>
                    </sml:ComponentList>}
                 </sml:components>
         </sml:System>
   </sml:member>
</sml:SensorML>
```

Figura 3.6: Exemplo parcial de dados em SensorML

sensores virtuais pensamos criar uma instância da máquina virtual de Callas por sensor, de forma semelhante ao que acontece numa rede real, em que cada dispositivo possui uma máquina virtual. Após ser analisada a estrutura da máquina virtual verificou-se que bastava criar uma instância da máquina virtual de Callas e várias instâncias do seu motor de execução, pois é este o elemento que permite manter o estado de cada sensor. Graças a esta optimização foi possível economizar recursos na criação de cada sensor virtual.

De forma a concretizar a criação de vários motores de execução foi necessário alterar o código do motor de execução, tornando essa classe numa *thread*. Assim, quando é recebido código para a reprogramação de dispositivos, é também enviado para a MuFFIN-CVM uma lista com os identificadores de cada sensor, sendo criado um número de *threads* igual ao número de identificadores existente na lista. É no momento da criação de cada *thread* que lhe é indicado qual o identificador do sensor que vai representar, para que possa ler da base de dados as observações feitas pelo seu homólogo na rede real.

Simular as leituras dos sensores: durante a execução de cada sensor pode haver a necessidade deste interagir com o sistema operativo ou com o *hardware*, como por exemplo, ler uma temperatura. Dado que não é possível a um sensor virtual obter um valor diretamente do sensor físico, alteramos a interface que permite realizar as chamadas ao sistema de forma a ser realizada uma leitura à base de dados. Assim, quando um sensor virtual realiza uma observação é feito um pedido à base de dados com o seu identificador, sendo-lhe devolvido o valor da última observação do sensor real com aquele identificador. As operações suportadas pela máquina virtual são definidas por um ficheiro, como o representado na Figura 3.7.

De forma a manter os valores da base de dados atualizados, a rede real continua a enviar os dados das suas leituras para a base de dados.

```
defmodule Nil: pass

defmodule Extern:
  bool logLong(long val)
  bool logDouble(double val)
  bool logString(string val)
  long macAddr()
  long getLuminosity()
  double getTemperature()
  long getTime()
  long getBatteryLevel()
```

Figura 3.7: Exemplo do ficheiro que define a assinatura das operações a serem implementadas na máquina virtual

Simular a comunicação entre sensores: para simular o fluxo de mensagens trocadas, implementamos um vetor circular onde cada mensagem enviada contém a seguinte informação:

- **payload:** informação enviada pelo sensor;
- **idSender:** identificador do sensor que envia a mensagem. Este identificador garante que a mensagem não é entregue ao emissor.
- **idMessage:** identificador da mensagem, que evita que cada sensor leia mensagens repetidas;
- **port:** porto a que se destina a mensagem, de forma a modelar diferentes canais de comunicação.

Como mostra a Figura 3.8 optámos por um vetor circular para simular o caso em que existem perdas de mensagens, pois caso haja um grande número de mensagens na "rede" é possível que alguma mensagem seja substituída antes de todos os sensores a terem conseguido ler. A dimensão desse vetor é determinada pelo número de sensores pertencentes à rede, por exemplo numa rede constituída por 10 sensores o vetor circular terá 10 posições.

Por fim foi necessário alterar as interfaces de rede (entrada e saída) da máquina virtual para estas acederem ao vetor circular implementado.

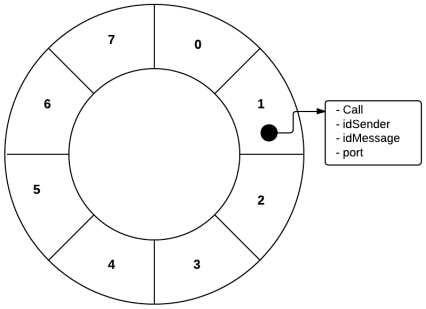

Figura 3.8: Esquema de mensagens

3.3 Considerações finais

Neste capítulo apresentamos o conceito que permite realizar a reprogramação remota de todo o tipo de dispositivos, independentemente das suas capacidades, e com recurso a apenas uma linguagem. Como prova de conceito desse modelo integramos a máquina virtual de Callas no MuFFIN. Com esta integração criamos o componente MuFFIN-CVM, de forma a permitir a execução de código ao nível do *middleware*. Nesta integração tivemos como foco tornar a utilização do serviço de reprogramação o mais homogéneo possível, pelo que optámos por usar o mesmo serviço de reprogramação para redes físicas e virtuais, não sendo possível aos clientes identificar o tipo de rede com que estão a interagir. Na Figura 3.9 pode-se observar a arquitetura do MuFFIN já com o novo componente integrado.

No decorrer do desenvolvimento da nossa abordagem identificamos três problemas principais: 1) a representação dos sensores, que foi resolvido através da criação de um motor de execução da máquina virtual em representação de cada sensor, 2) a simulação das leituras dos sensores, que foi solucionado recorrendo aos valores enviados pela rede real para a base de dados e 3) a simulação da comunicação entre sensores virtuais, que foi resolvida através da criação de um sistema que simula a rede onde fluem as mensagens trocadas.

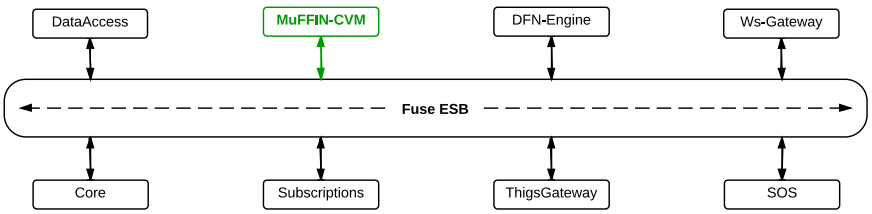

Figura 3.9: Arquitetura do MuFFIN

Capítulo 4

Comunicação entre redes de sensores

Neste capítulo é apresentada a nova componente do MuFFIN que permite realizar a comunicação entre redes de sensores distintas ao nível do *middleware*, ao invés dessa informação ser tratada pelas aplicações cliente. É também apresentada a concretização desta funcionalidade com o MuFFIN, sendo realizada uma descrição de cada um dos elementos que fazem parte deste componente e quais os seus detalhes de implementação.

4.1 Comunicação entre redes de sensores ao nível do *middleware*

Existem muitos sistemas que utilizam pelo menos dois tipos de redes de dispositivos, onde uma é responsável por monitorizar um determinado elemento, por exemplo a temperatura, e a outra é responsável por atuar de forma a controlar esse elemento, por exemplo aumentar ou diminuir a potência do ar condicionado para manter uma determinada temperatura.

Utilizando um *middleware* para estabelecer a comunicação entre uma rede de sensores de monitorização e outra de atuadores, teria que existir interação com os dados enviados ao nível aplicacional (aplicações cliente do *middleware*) para aí serem transformadas de acordo com as necessidades da rede de atuadores e posteriormente enviadas para a rede de destino.

De forma a optimizar este tipo de interação, foi idealizado um sistema que permite estabelecer a comunicação entre redes de sensores ao nível do *middleware*, deixando de haver necessidade de que essa informação flua até às aplicações cliente, ficando no *middleware* a responsabilidade de realizar o encaminhamento e adaptação da informação enviada pelas redes de sensores, como mostra o exemplo da Figura 4.1, onde uma determinada rede origem (Rede 1) envia informação para outra rede (Rede 3).

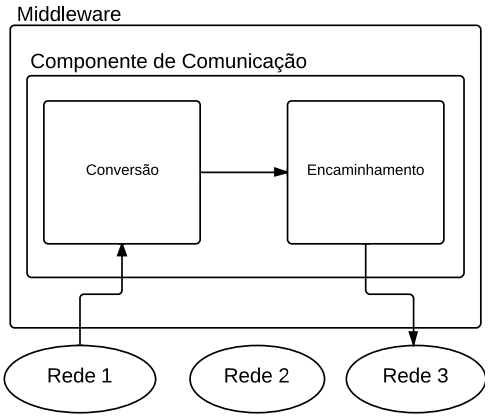

Figura 4.1: Esquema conceptual de comunicação entre redes de sensores

4.2 *Things Gateway Communication*

Como prova de conceito de comunicação de redes de sensores ao nível do *middleware* criamos o componente *Things Gateway Communication*. Este é um componente que permite encaminhar e adaptar a informação de uma rede origem para uma de destino. Graças a esta nova funcionalidade foi-nos possível reduzir o número de mensagens trocadas entre o *middleware* e as aplicações cliente, no contexto da comunicação entre redes.

4.2.1 Elementos do componente *Things Gateway Communication*

Esta secção apresenta os elementos constituintes do *Things Gateway Communication* e a forma como é possível estabelecer a comunicação entre duas redes de sensores.

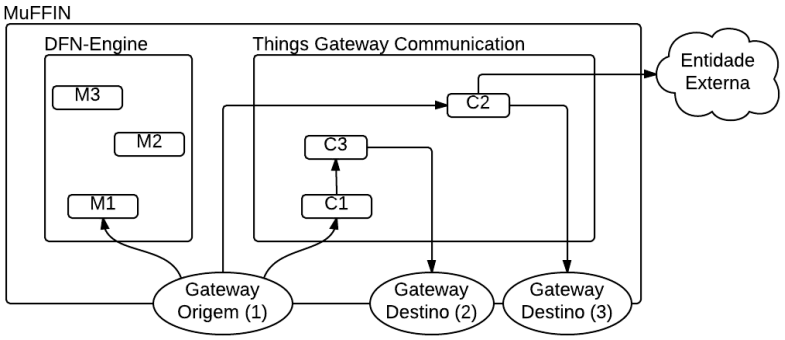

Figura 4.2: Componente ThingsGateway-Communication

A Figura 4.2 mostra que o *Things Gateway Communication* é constituído por conversores (na Figura 4.2 representados por C1, C2 e C3). Estes conversores recebem a informação dos *gateways*, realizam as transformações necessárias e encaminham a informação para o destino. Na Figura também é possível observar que é possível invocar serviços de uma entidade externa ao MuFFIN. Seguidamente apresentamos a forma de funcionamento dos conversores e da entidade externa.

Conversores

De forma a suportar a comunicação entre redes ao nível do *middleware*, foi desenvolvido um sistema de conversores de protocolos. Este permite adaptar os protocolos das redes de origem e destino, em que os dados recebidos num *gateway*, segundo o protocolo da rede de origem, são adaptados de acordo com o protocolo da rede de destino.

Para resolver o problema de conversão de protocolos entre as redes optamos por estender o MuFFIN de forma a permitir a instalação de conversores, criando uma solução flexível que permite reutilizar conversores existentes e realizar a composição de conversores. Por exemplo, podemos criar um sistema de conversão da rede A para a rede C por composição dos conversores da rede A para a rede B e da rede B para a rede C, tal como é possível observar na figura 4.3.

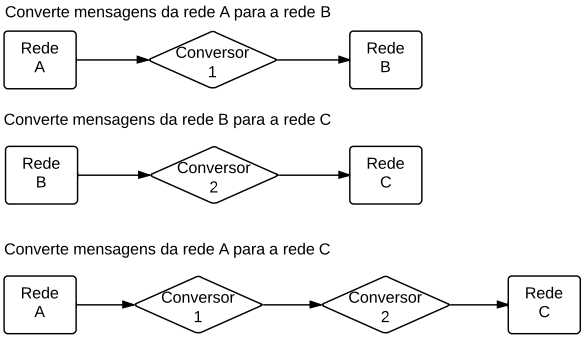

Figura 4.3: Exemplo de composição de conversores

Estes conversores são classes desenvolvidas na linguagem Java e que implementam a interface *CommunicationConverter*, de forma a que o MuFFIN as possa utilizar. Nesta funcionalidade é utilizado o padrão *Adapter* [18], pois os conversores possuem uma interface comum mas cada um tem um comportamento diferente. Após o utilizador desenvolver um conversor, tem de utilizar o serviço *web* destinado à instalação de conversores no MuFFIN. Na secção seguinte descrevemos a forma como funciona o sistema de conversores após a sua instalação.

Interligação de conversores

Após a instalação dos conversores, é necessário associar os *gateways* das redes origem e destino e quais os conversores envolvidos. A associação entre conversores e *gateways* é definida através de um documento XML (ver exemplo na Figura 4.4). Este documento inclui os identificadores dos *gateways* de origem e destino e os conversores utilizados. Adicionalmente pode ser definido o *Uniform Resource Identifier* (URI) de uma entidade externa cuja função será explicada na secção seguinte.

Para os conversores especificados no XML serão criadas instâncias para cada um deles, as quais estão interligadas pelo seu identificador, de forma a ser estabelecida uma sequência entre os conversores envolvidos. Esta interligação é garantida pelo serviço de publicação/subscrição oferecido pelo ActiveMQ.

Na Figura 4.4 apresentamos um exemplo de um ficheiro XML que associa os *gateways* 1 e 2, aos conversores 3 e 1 (o esquema XML deste documento pode ser consultado no anexo A.1).

```
<?xml version="1.0" encoding="UTF-8" ?>
<deploy>
   <description>Exemplo de um conversor</description>
   <instanceSource id="1"/>
   <instanceDest id="2"/>
   <converters>
          <converter serviceId="3"/>
          <converter serviceId="1"/>
   </converters>
   <externalService> http://www.lasige.fc.ul.pt/externService </externalService>
 </deploy>
```

Figura 4.4: Exemplo de um XML de associação

A comunicação estabelecida entre os *gateways* não se resume apenas a uma comunicação ponto-a-ponto, ou seja, pode existir comunicação de um para muitos. No caso da Figura 4.2 existe, por exemplo, uma comunicação de um para muitos, em que o *Gateway Origem 1* envia informação para o *Gateway Destino 2* e *Gateway Destino 3*.

De forma a ilustrar o funcionamento deste mecanismo, vamos considerar um caso em que é necessário monitorizar a temperatura de um *datacenter*. Para essa tarefa dispomos de uma rede que monitoriza a temperatura e outra que controla o sistema de ar condicionado, sendo necessário estabelecer comunicação entre as duas redes. Vamos considerar que a rede de temperatura comunica com o *middleware* através do *Gateway Origem 1* e a rede de ar condicionado comunica com o *Gateway Destino 2*. Para que seja possível à rede que mede a temperatura enviar informação para a rede que atua sobre o ar condicionado, é necessário que sejam instalados e instanciados os conversores C1 e C3 (ver XML da Figura 4.4), que adaptam os protocolos utilizados pelas redes, sendo executado o caminho *Gateway Origem 1 - C1 - C3 - Gateway Destino 2*, sempre que a rede de sensores envie uma mensagem para a rede de atuadores.

A comunicação realizada entre os *gateways* de origem, os conversores e os *gateways* de destino é estabelecida segundo o paradigma de publicação/subscrição. Assim, os conversores subscrevem a informação enviada pelas redes de origem e após o seu processamento publicam-na de forma a ser lida e processada por outro conversor ou lida pelo *gateway* de destino.

Entidade Externa

Podem existir casos em que não é possível converter os dados recebidos de uma rede com recurso aos conversores instalados. Para realizar essa tarefa existe um mecanismo que permite utilizar uma entidade externa, o qual possibilita a invocação de um serviço que realize a conversão. Por exemplo, uma rede pode trocar mensagens num formato *byte-code* específico da linguagem de programação (*e.g.*, a linguagem Callas ou Java). Se se pretender converter mensagens de, ou para, uma rede deste tipo, é necessário ter partes do compilador ou da máquina virtual (por exemplo, para fazer análise sintática ou síntese de *bytecode*) que não estão disponíveis ao nível do *middleware*. O mecanismo que propomos oferece várias vantagens, em particular, não sobrecarrega o MuFFIN com *software* e configurações adicionais, facilita a instalação e manutenção do *middleware* e proporciona uma forma elegante de abstrair o mecanismo de conversão, quer ao nível do compilador da linguagem usada, quer do sistema operativo ou outros requisitos necessários à sua execução. Esta entidade é especificada por um URI, quando se instância um conversor, havendo apenas a necessidade do utilizador garantir que a entidade especificada existe e que faz a transformação que é pretendida.

4.2.2 Novos Serviços

A componente *Things Gateway Communication* levou à criação de serviços *web* para disponibilizar as novas funcionalidades do *middleware* às aplicações de alto nível. Seguidamente descrevemos e esquematizamos o funcionamentos desses novos serviços.

***deployConverter* (Adicionar conversor):** permite que os clientes possam instalar conversores de mensagens. Os argumentos deste serviço são: o nome da classe do conversor, uma descrição e código fonte. Este serviço cria uma nova entrada na base de dados de forma a guardar a informação do módulo, sendo devolvido um identificador da entrada criada.

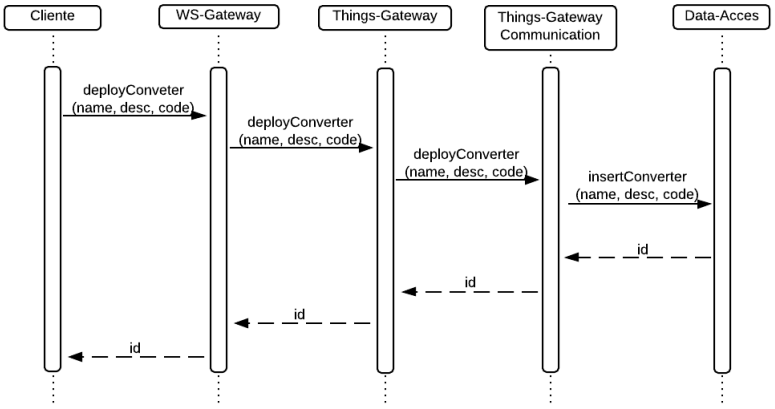

Figura 4.5: Diagrama de sequência do serviço adicionar conversor

***instantiateConverter* (Instanciar conversor):** Este serviço utiliza o XML de instanciação descrito na secção 4.2.1. Ao ser recebido um pedido, o código é compilado, são criadas as dependências descritas no XML (caso existam) e é adicionada uma entrada na base de dados com os parâmetros da instanciação, sendo devolvido o identificador dessa entrada. Também é compilado o código do conversor, sendo que este é instanciado de acordo com os argumentos especificados no XML de instanciação (Figura 4.4).

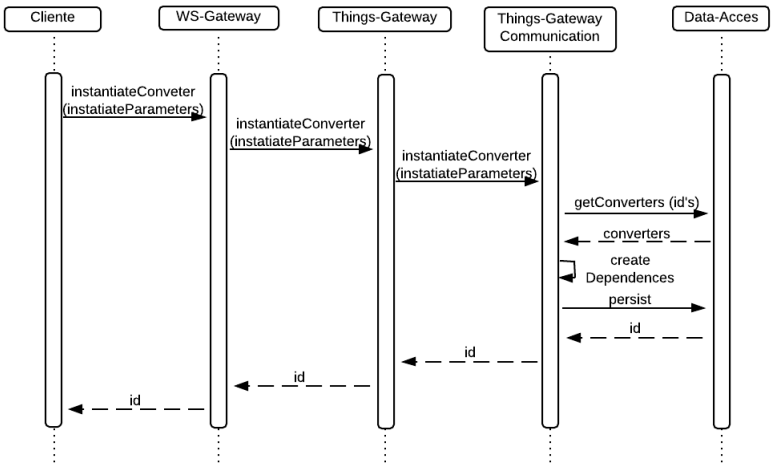

Figura 4.6: Diagrama de sequência do serviço instanciar conversor

***removeConverter* (Eliminar conversor):** elimina um conversor caso este não esteja instanciado ou faça parte das dependências de uma instanciação. Para tal, apenas é necessário indicar qual o identificador do módulo. Quando o serviço é utilizado é removida a entrada da base de dados referente ao identificador do conversor, sendo devolvida a informação de se o conversor foi removido.

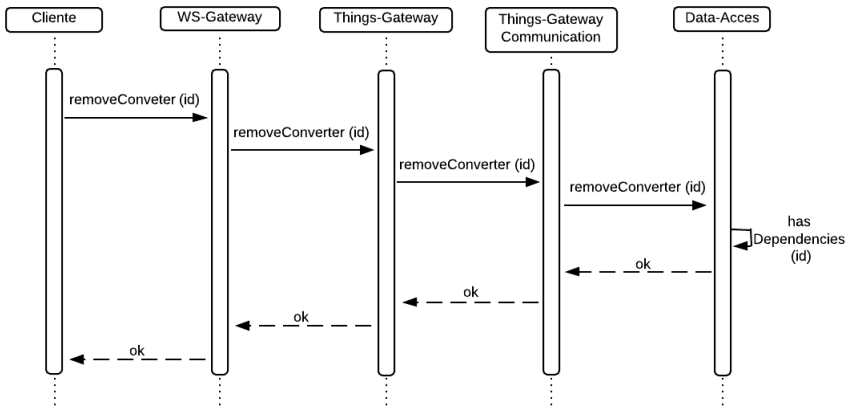

Figura 4.7: Diagrama de sequência do serviço remover conversor

***removeConverterInstance* (Eliminar instância de conversor):** remove uma instância de um conversor, sendo necessário indicar o identificador da instância a eliminar. Ao ser utilizado este serviço todas as instâncias indicadas e as suas dependências serão eliminadas, bem como as respetivas entradas na base de dados, sendo indicado ao cliente se o conversor foi removido.

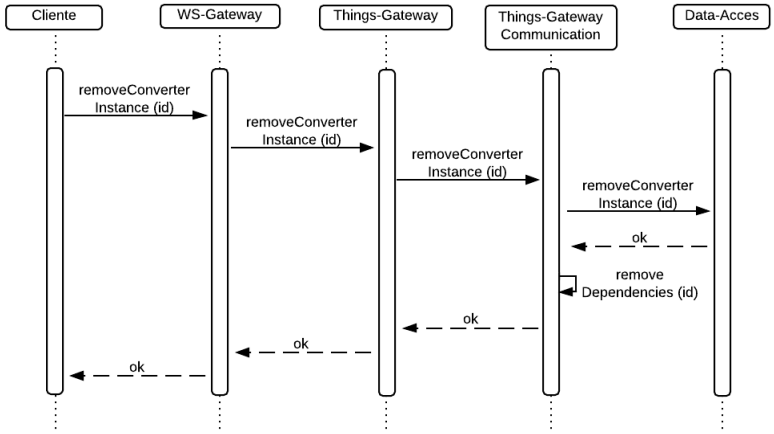

Figura 4.8: Diagrama de sequência do serviço remover instância

***getCommunicationComponentsInfo* (Obter informações):** obtém informação dos conversores e instâncias existentes. Assim, quando o serviço é invocado, é enviado para os clientes um XML com as informações dos conversores e instâncias. Nos anexos B.1 e B.2 é possível observar o esquema XML que define a estrutura das respostas envidas por este serviço e um exemplo de uma resposta obtida pela invocação deste serviço, respetivamente.

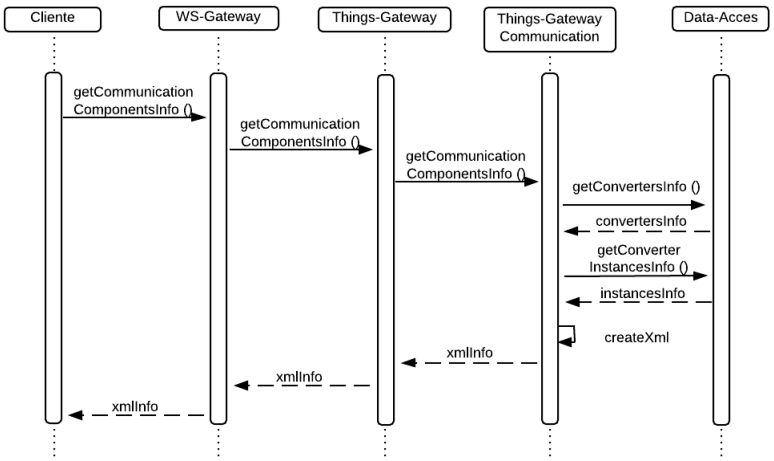

Figura 4.9: Diagrama de sequência do serviço obter informações

4.3 Implementação

A implementação deste componente, tal como os já existentes, seguiu a aquitetura OSGi de forma a possibilitar a integração entre componentes. Para este componente foi criada a interface representada na Figura 4.10, que disponibiliza os serviços deste componente, de forma a estes poderem ser disponibilizados segundo serviços *web* pelo componente *WS-Gateway*.

```
public interface IThingsGatewayCom {

        public int deployCommunicationModule(String name, String desc, String code);

        public int instantiateCommunicationModule(String instatiateParameters);

        public boolean removeConverterModule(int id);

        public boolean removeCoverterInstance(int id);

        public String getCommunicationComponentsInfo();

}
```

Figura 4.10: Interface do componete *Things Gateway Communication*

Na secção 4.2.1 vimos que os *gateways* e os conversores interagem seguindo um sistema de publicação/subscrição. Para implementarmos esse sistema recorremos à utilização da ferramenta ActiveMQ, um serviço integrado no FuseESB, através do qual os diversos elementos de comunicação subscrevem e publicam informação.

Os conversores são classes escritas na linguagem Java (Figura 4.11), desenvolvidas pelos utilizadores e têm que estender uma classe específica (CommunicationConverter), pois é esta que possui o métodos capazes de iniciar os serviços de publicação/subscrição que permitem subscrever e publicar informação.

```
public class TestCommunicationModule extends CommunicationConverter {

        public TestCommunicationModule() {
                super();
        }
        @Override
        public void processInfo(byte [] recvdObj) {

                //TO DO

        }
}
```

Figura 4.11: Estrutura de um conversor

Pelo XML da Figura 4.4, que tem como função instanciar os conversores podemos verificar que existe uma relação de sequência entre os conversores. Assim o primeiro conversor (3) é o que faz a última transformação e publica a informação para o *gateway*

de destino, o último conversor (1) é o que subscreve o *gateway* de origem, recebendo a informação deste e processando a primeira alteração. No caso em que existem mais conversores, eles publicam e subscrevem informação de acordo com a ordem em que aparecem.

4.3.1 Alterações aos componentes existentes

Foram feitas alterações nos componentes *DFN-Engine* e *ThingsGateway*, de forma a melhorar a interação com o utilizador. Assim, foram realizadas alterações aos seguintes componentes:

ThingsGateway: foi necessário alterar os *gateways* de maneira a que estes possam também subscrever tópicos, de forma a serem notificados quando uma rede envia informação para outra.

DataAccess: foi necessário criar as classes que representam as entidade de acesso às tabelas que foram criadas devido à extensão realizada ao MuFFIN, sendo também criadas classes que permitem fazer consultas mais específicas aos dados existentes na base de dados.

WS-Gateway: foram implementados os serviços *web* que permitem disponibilizar os novos serviços do MuFFIN aos clientes.

4.3.2 Modelo de dados

Na Figura 4.12 encontra-se uma versão simplificada do modelo de base de dados do *middleware*. Nesta representação foram omitidas algumas tabelas, pelo que é possível observar as entidades mais importantes da base de dados, nomeadamente as entidades que dizem respeito às observações dos sensores e as entidades que guardam a informação sobre os *gateways* utilizados para interagir com as redes. Também é possível observar quais as que surgiram devido à extensão do MuFFIN (destacadas a verde), as quais são explicadas seguidamente.

Com o desenvolvimento do componente (*ThingsGatewayCommunication*), há necessidade de criar três novas tabelas: a tabela *communication_module*, que armazena a informação dos conversores que são instalados no MuFFIN; (2) a tabela *communication_instance*, que guarda informação dos conversores que são instanciados. Esta estabelece uma ligação com a tabela *communication_module* para se identificar qual o conversor que está a ser instanciado e estabelece outra ligação com a tabela *service_instance* de forma a ser possível obter a informação dos *gateways* das redes que vão comunicar; (3) por fim a tabela

communicanion_dependencies, que armazena os identificadores dos conversores que cada instância depende.

A Figura 4.12 ilustra também, a tabela *gateway_type* (destacada a vermelho). Esta tabela foi criada no contexto da funcionalidade descrita no Capítulo 3, e tem como função armazenar a informação sobre o tipos de *gateway* e a informação SensorML associada a cada um destes elementos.

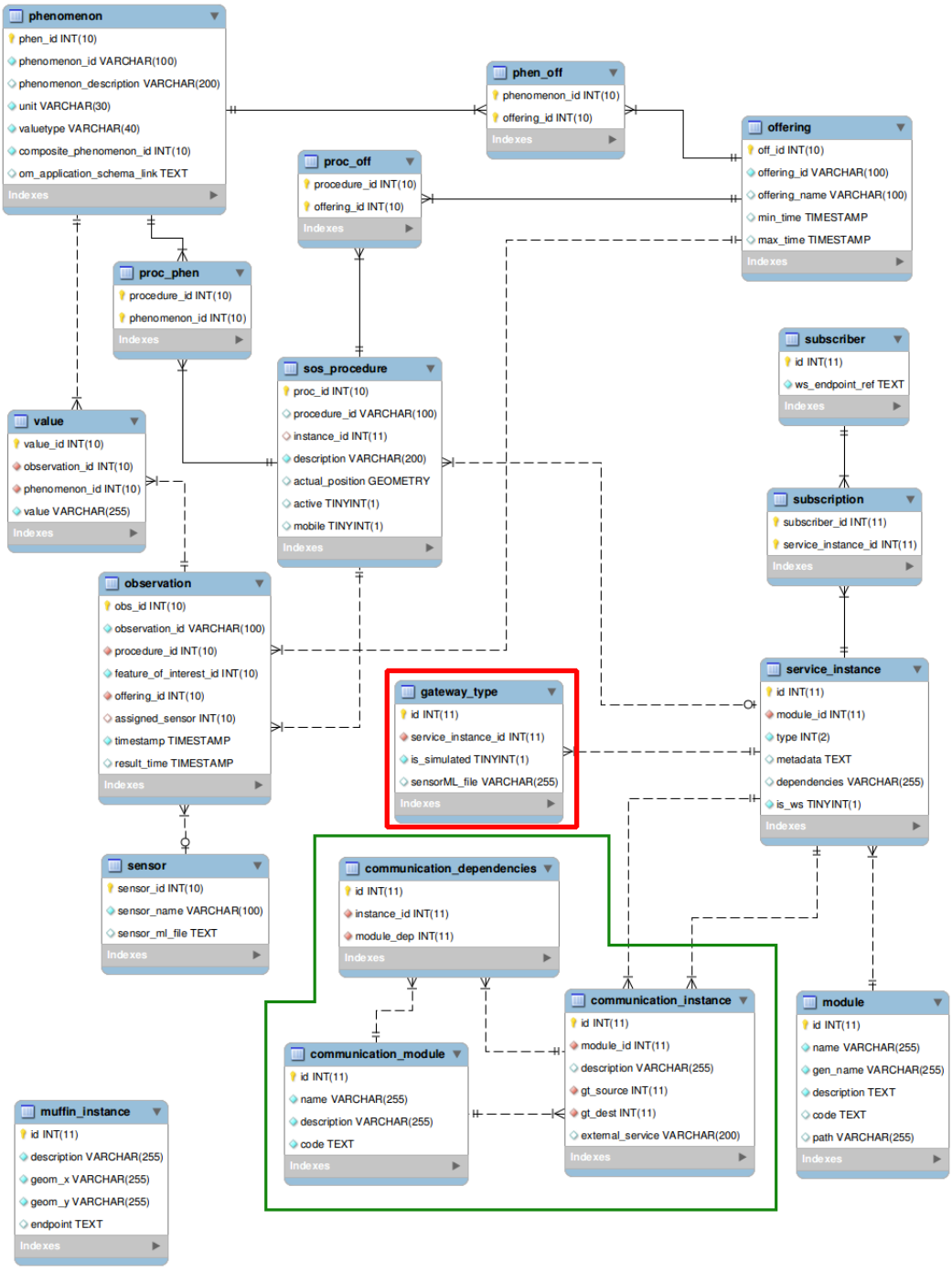

Figura 4.12: Versão simplificada do modelo de dados do MuFFIN

4.4 Considerações finais

Na Figura 4.13 encontra-se a verde o novo componente do MuFFIN apresentado neste capítulo, e a vermelho o componente MuFFIN-CVM apresentado no Capítulo 3. Neste capítulo apresentamos o componente *ThingsGatewayCommunication*, o qual permite estabelecer a comunicação entre redes distintas. No capítulo foram também apresentados os detalhes de implementação do componente e foram explicados os serviços que o MuFFIN passou a incluir, sendo eles os seguintes: instalar conversor, instanciar conversor, remover conversor, remover instância de conversor e obter informação dos conversores.

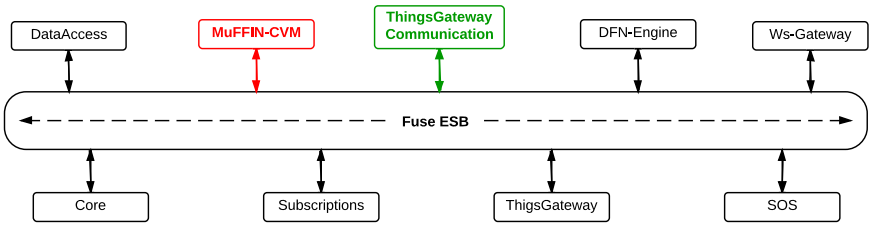

Figura 4.13: Arquitetura do MuFFIN

Capítulo 5

Avaliação

Neste capítulo apresentamos o ambiente em que foram realizados os testes. São também apresentados e discutidos os resultados obtidos nos testes realizados aos novos componentes da extensão do MuFFIN.

5.1 Ambiente de testes

O *middleware* encontra-se instalado no servidor aplicacional Fuse ESB. Estes serviços estão a correr numa máquina virtual criada no Virtual Box (versão 4.1.22), com as seguintes características:

- Processador: Intel(R) Core(TM)2 Duo E8400 @ 3.00GHz (1 CORE)
- Memória RAM: 1000 Mb
- Sistema Operativo: Linux, Ubuntu 12.04 Server

Nesta secção são apresentados os resultados dos testes ao desempenho dos componentes desenvolvidos, logo não serão tidos em conta os tempos que as mensagens demoram a chegar de uma rede de sensores a um *gateway* e vice-versa.

Os testes realizados ao componente MuFFIN-CVM compreendem a medição do tempo de arranque de uma rede de sensores (período entre a chegada do código Callas até à execução do sensor iniciar) e a medição do tempo de arranque mais o processamento de uma leitura da base de dados (chamada ao sistema operativo). Em ambos os casos executamos redes de sensores com diferentes dimensões (10, 50, 100, 200, 500, 1000, 1500 e 2000 nós), de forma a ser possível verificar qual o custo a nível de tempo de processamento relativamente ao número de sensores de uma rede. Para este componente foram também realizados testes de forma a medir a memória consumida pelo MuFFIN ao serem executadas redes com diferentes dimensões.

Para avaliar o componente *ThingsGateway-Communication* usamos três tipos de testes: (1) as redes usam a mesma linguagem/protocolo e as mensagens são enviadas diretamente de um *gateway* para outro; (2) as redes usam a mesma linguagem/protocolo e as

mensagens são enviadas através de um conversor identidade que apenas as encaminha, não havendo necessidade de adaptação de mensagens; (3) um teste em que é necessário realizar a conversão de uma *string* num valor inteiro.

Tendo em conta que apenas pretendemos avaliar o desempenho das extensões realizadas ao MuFFIN, não foram realizados testes que incluíssem a entidade externa.

5.2 Testes realizados ao componente MuFFIN-CVM

Sensores	10	50	100	200	500	1000	1500	2000
Arranque (ms)								
Média	11.08	33.22	64	130.73	351.38	795.66	1167.1	1625.01
D.Padrão	9.99	5.07	3.86	2.83	5.68	5.58	5.55	4.14
Arranque e processamento (ms)								
Média	20.69	46.22	78.69	160.56	450.96	955.37	1448.15	2025.14
D.Padrão	9.37	6.91	5.61	3.59	3.03	2.57	2.56	2.90
Memória Consumida (MBytes)								
Média	52.42	54.47	58.18	64.18	81.52	83.47	89.57	90.50
D.Padrão	1.35	1.2	1.59	2.72	4.72	3.96	4.07	3.07

Tabela 5.1: Valores dos testes realizados

No gráfico representado na Figura 5.1 é possível observar o desempenho do componente MuFFIN-CVM, no que diz respeito à simulação do arranque de uma rede de sensores com vários nós e à simulação do arranque de uma rede em que também é realizada uma leitura da base de dados. Por exemplo, numa rede com 10 nós, após os sensores arrancarem, são realizados 10 pedidos à base de dados, um por sensor. Para cada uma das redes executadas foram realizados 100 medições, cujos valores médios e de desvio padrão obtidos se encontram representados na Tabela 5.1.

Pela observação do gráfico da Figura 5.1 podemos verificar que, quando são executadas redes com poucos nós (até 200), os tempos de arranque e de arranque e execução, são muito próximos. Já quando são executadas redes com um número de nós superior (a partir de 500) podemos verificar que o tempo de arranque e processamento começa a ser um pouco mais elevado do que o tempo só de arranque. No entanto em ambos os casos o crescimento é linear.

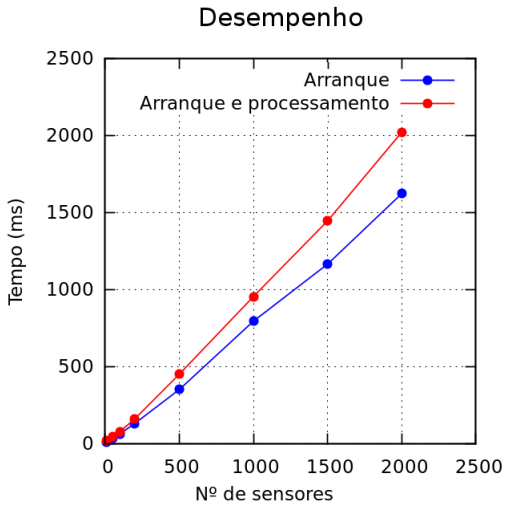

Figura 5.1: Desempenho do arranque de sensores

Relativamente aos testes realizados para avaliar o consumo de memória, representados no do gráfico da Figura 5.2, é possível observar que quando são executadas redes com uma dimensão de até 500 sensores, existe um crescimento mais significativo da memória consumida pelo MuFFIN. Quando são executadas redes com mais de 500 sensores podemos verificar que a memória consumida não cresce de forma tão acentuada, o que nos permite dizer que o sistema de execução de sensores é escalável. Ainda assim, estes valores podem não ser muito representativos, pois esperávamos um gráfico com crescimento linear e não o verificado. Apontamos como razão destes resultados o facto de não controlarmos totalmente o mecanismo de gestão de memória do Java (*Garbage Collector*), uma vez que apenas forçávamos que este actuasse no final de cada medição.

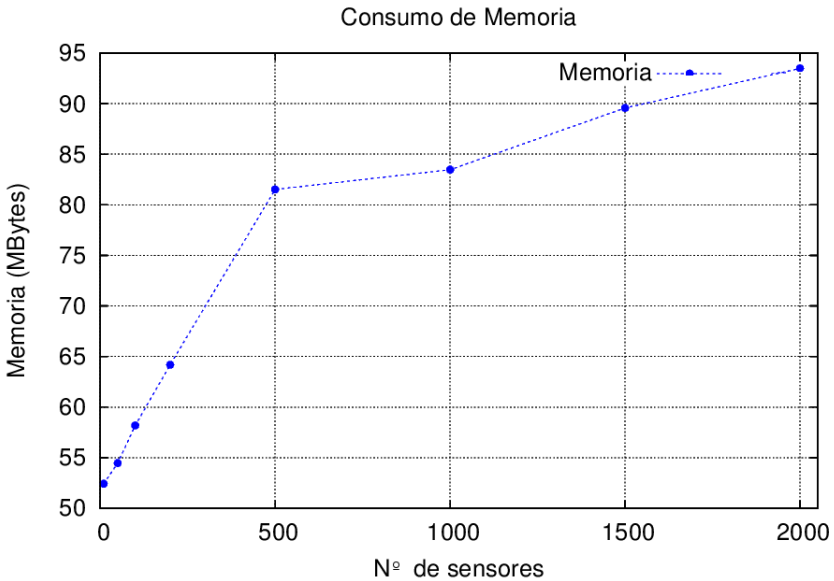

Figura 5.2: Consumo de memória

5.3 Testes realizados ao componente *Things Gateway Communication*

No gráfico da Figura 5.3 encontram-se representados os resultados obtidos nos testes realizados ao componente de troca de mensagens entre *gateways*. Nos três casos foram realizadas 100 medições chegando-se aos valores representados na Tabela 5.2.

Testes	S/ adaptaçao	C/ identidade	C/ adaptaçao
Média (ms)	11.5	11.8	12.2
D.Padrão	1.14	1.13	1.17

Tabela 5.2: Valores dos testes realizados

Pelos valores obtidos podemos verificar que, o custo de enviar mensagens diretamente para o *gateway* (s/ adaptação) é pouco inferior em relação à utilização do componente de identidade, não existindo perdas de tempo significativas no encaminhamento das mensagens. O valor obtido no envio com adaptação não é muito superior aos outros valores, no entanto este valor depende do tipo de adaptação realizada, visto que neste caso se tratava de uma adaptação que não necessitava de muito processamento (converter uma *string* num inteiro).

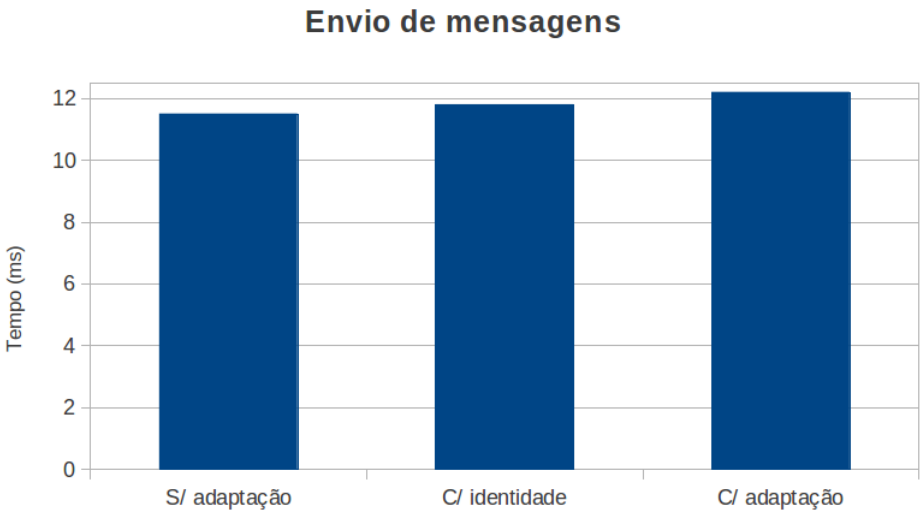

Figura 5.3: Tempo de envio de mensagens

5.4 Simulação de situações em ambiente real

Nesta secção é apresentado um cenário de utilização do MuFFIN em que possuímos uma rede se sensores não reprogramável e uma rede de atuadores. Neste cenário pretende-se que a rede de sensores seja reprogramada e envie informação para a rede de atuadores.

Para a concretização deste cenário apresentamos também uma possível forma de interação com o MuFFIN, através da execução de processos de negócio, permitindo criar a infraestrutura necessária à concretização do cenário apresentado.

5.4.1 Cenário de utilização

O cenário utilizado para ilustrar o nosso trabalho tem como objetivo o controlo da temperatura e da humidade de uma sala de arquivo de documentos antigos. Para tal dispomos de uma rede que realiza a monitorização da temperatura e da humidade da sala e uma rede de atuadores que permite controlar os equipamentos de ar condicionado que ajustam a temperatura e a humidade do ambiente.

A aplicação de monitorização define o comportamento dos sensores de modo a ser notificada em intervalos de tempo regulares. De maneira a não haver notificações de forma periódica para o sistema de controle do ar condicionado, pretende-se alterar o comportamento dos sensores de modo a que estes enviem notificações apenas quando os valores da temperatura estiverem fora do intervalo de 16ºC a 20ºC ou os da humidade fora do intervalo de 45% a 60%, fazendo com que o sistema de ar condicionado atue apenas quando os valores do intervalo são ultrapassados. Para tal os sensores necessitam de ser reprogramados. Com a extensão do MuFFIN apresentada, esta reprogramação pode ser efetuada independentemente das capacidades dos sensores.

5.4.2 Descrição da experiência

Para ilustrar o nosso cenário de utilização, utilizamos dispositivos SunSPOT para de ler os valores de temperatura de forma periódica. Como estes dispositivos não permitem realizar a leitura de valores de humidade, apenas serão considerados os valores de temperatura.

Em representação dos atuadores sobre os equipamentos de ar condicionado utilizamos também dispositivos SunSPOT, que no caso em que é necessário atuar sobre o ar condicionado, estes acendem os seus *led's*.

De forma a interagir com o *middleware* criamos um cliente *web* para o MuFFIN. Este cliente tem por base a definição de processos de negócio, utilizando a linguagem BPEL. Estes processos são responsáveis por realizar cada uma das interações com o *middleware*. Desta forma, e para conseguir desenvolver o sistema necessário ao nosso cenário, definimos os processos de negócio necessários à interação com o MuFFIN.

De forma a criar uma configuração no MuFFIN capaz de satisfazer a situação apresentada no nosso caso de utilização, foi necessário orquestrar os processos de negócio anteriormente definidos. Assim, o primeiro processo de negócio a ser executado foi o de instalação de um *gateway* (Figura 5.4), de maneira a instalar um *gateway* capaz de interagir com a rede de dispositivos SunSPOT, responsável por ler as temperaturas. Após a instalação foi devolvido o identificador atribuído ao *gateway* instalado. Neste processo os

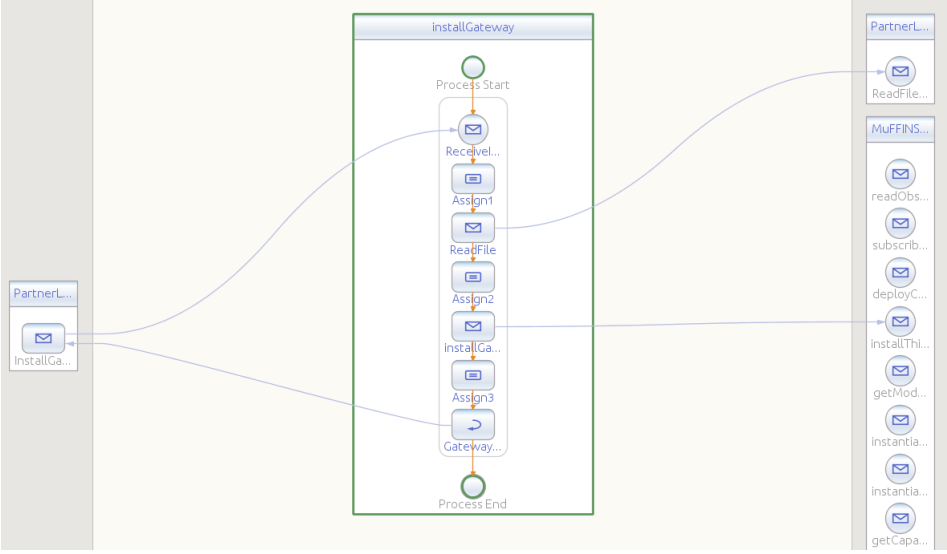

Figura 5.4: Processo BPEL para instalar um *gateway*

elementos BPEL mais importantes são o *ReadFile*, que é responsável por carregar para memória o *gateway* a ser instalado, e o *installGateway* que é o responsável por chamar o serviço *web* que permite a instalação de um *gateway*.

Após a instalação do *gateway*, instalou-se um módulo no DFN-Engine (Figura 5.5), sendo devolvido o identificador que lhe foi atribuído. Este módulo tem como objetivo servir de módulo identidade, em que este subscreve o *gateway* da rede que lê as temperaturas, armazenando essa informação na base de dados. Para que fosse possível ao módulo instalado assumir a função de módulo identidade do *gateway* da rede que monitoriza a temperatura foi necessário realizar a sua instanciação. Para tal executou-se o processo de negócio da Figura 5.5, onde foi especificado o XML de instanciação do módulo no qual consta o identificador, devolvido após a instalação do módulo, e o identificador, devolvido após a instalação do *gateway*, permitindo que o módulo seja capaz de subscrever a informação enviada pelo *gateway*. Nestes dois processos de negócio os dois elementos BPEL responsáveis por realizar as chamadas aos *web services* do MuFFIN são o *DeployModule* e *InstantiateModule* que permitem instalar e instanciar um módulo no componente DFN-Engine.

Como pretendemos reprogramar a rede de sensores de forma a que esta apenas envie os valores de temperatura quando estes estiverem fora do intervalo 16ºC a 20ºC, em vez de enviar valores periodicamente, foi necessário instalar um *gateway* virtual, pois não é possível reprogramar os dispositivos SunSPOT com a linguagem Callas, visto que no nosso caso os dispositivos não possuíam a máquina virtual de Callas instalada. Desta forma usamos novamente o processo representado na Figura 5.4 para instalar o *gateway* virtual. Nesta instalação foi enviado um documento SensorML a descrever a rede de sensores com a qual o *gateway* irá comunicar (neste caso uma descrição igual à da rede

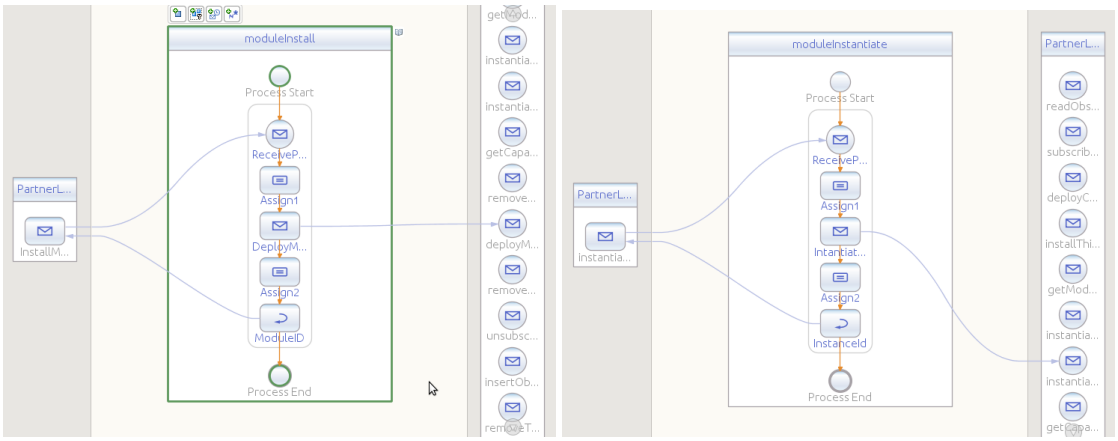

Figura 5.5: Processos BPEL para instalar e instanciar um módulo

de dispositivos SunSPOT que utilizamos para monitorizar a temperatura). Seguidamente usamos o processo de negócio da Figura 5.6 para enviar o código Callas a ser executado ao nível do *middleware*, tendo sido criada uma rede virtual com as características especificadas no documento SensorML associado ao *gateway*. Neste processo de negócio os elementos BPEL mais importantes são o *ReadCode*, que carrega o código Callas a ser instalado, e o *InstallCode* que é responsável por chamar o serviço *web* de reprogramação.

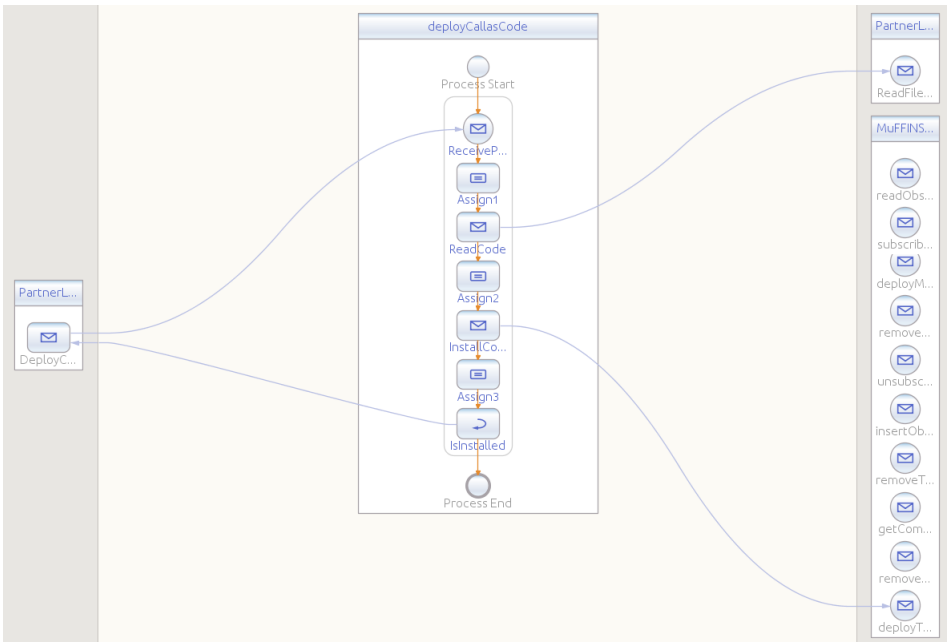

Figura 5.6: Processo BPEL para reprogramar rede

Após construída a parte de recolha de informação sobre as temperaturas, instalou-se um *gateway* para comunicar com a rede de atuadores, neste caso um dispositivo SunSPOT que apenas acende um *led* quando recebe a informação para atuar sobre o ar condicionado. Para tal, utilizamos mais uma vez o processo de negócio representado na Figura 5.4.

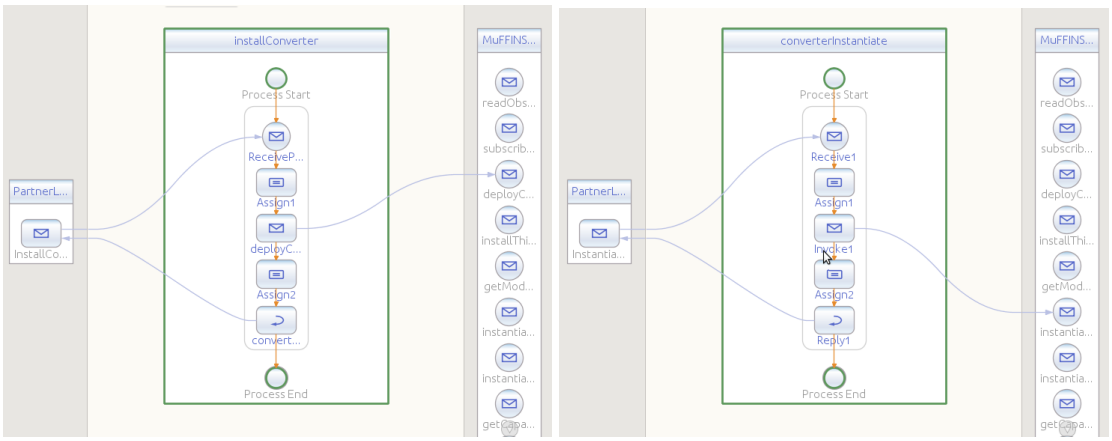

Figura 5.7: Processos BPEL para instalar e instanciar um conversor

Por fim, foram montados os últimos elementos, os quais permitiram estabelecer a comunicação entre a rede de sensores virtual e a rede de atuadores. Dessa forma, usamos os processos de negócio da Figura 5.7 para instalar e instanciar um conversor. Assim instalou-se um conversor que permite adaptar os valores enviados da rede virtual para valores que possam ser lidos pela rede de actuadores. Depois de instalado este módulo foi necessário realizar a sua instanciação, para isso enviou-se um XML onde foi especificado o identificador do módulo a ser instanciado e também os identificadores das redes de origem e destino.

Capítulo 6

Conclusão

Neste projeto apresentamos uma extensão ao *middleware* MuFFIN que permite generalizar a funcionalidade de reprogramação remota e o comportamento das redes de sensores, disponibilizando-a através de serviços *web*. Com este objetivo, incorporamos a máquina virtual do Callas no MuFFIN. Adicionalmente foi também desenvolvida uma componente que suporta a comunicação entre redes de sensores distintas.

Estas duas extensões realizadas ao MuFFIN fazem com que este *middleware* seja uma plataforma capaz de suportar a decomposição e distribuição de processos de negócio para as redes de sensores. De facto, convertendo a lógica de negócio que se pretende que seja executada numa rede de sensores para código Callas, é possível que esta seja enviada para as redes de sensores através do MuFFIN. Este tem ainda a capacidade de executar o código quando os sensores não incluem a máquina virtual de Callas. Nos casos em que o processo de negócio prevê a comunicação direta entre redes de sensores, o MuFFIN também assegura essa comunicação.

O estudo realizado sobre as tecnologias usadas pelo MuFFIN concluiu que este estava assente num conjunto de versões desatualizadas. Assim, fez-se uma pesquisa sobre as novas versões e procedeu-se à migração do MuFFIN para as novas versões das tecnologias. A migração do MuFFIN proporcionou melhoramentos a nível de segurança em algumas das tecnologias utilizadas, nomeadamente os mecanismos de autenticação do ActiveMQ e resolução de alguns erros existentes, como o de análise de XML existente no XML Beans, que anteriormente teve que ser resolvido manualmente através da substituição de ficheiros no núcleo do Fuse ESB, prática altamente desaconselhada.

Relativamente aos resultados obtidos podemos concluir que, no que diz respeito à execução de sensores ao nível do *middleware*, foi possível criar um sistema que pode ser escalável pois mesmo com aumento das dimensões das redes executadas, o desempenho é termos de tempo é linear. No caso da funcionalidade de comunicação entre redes de sensores podemos concluir que esta vem contribuir para uma forma mais simples de comunicação, visto que já não há necessidade das aplicações cliente lidarem com essa função quando se pretende estabelecer a comunicação entre redes, não existindo um custo

muito elevado associado ao processamento necessário ao nível do *middleware*.

O trabalho efetuado no âmbito deste projeto foi apresentado no simpósio de informática Inforum 2013, como uma comunicação de trabalho em curso.

Como trabalho futuro, pretendemos adequar o MuFFIN a correr em dispositivos com capacidade de computação limitada, como é o caso do Raspberry Pi, de forma a poder fazer parte de tecnologia a bordo de veículos e servir de interface a sistemas de transporte inteligentes (ITS).

Apêndice A

Instanciação de conversores

A.1 XSD de instanciação de conversores

```
<?xml version="1.0" encoding="UTF-8"?>
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
      <xs:element name="deploy">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="description" type="xs:string"></xs:element>
               <xs:element name="instanceSource">
                  <xs:complexType>
                     <xs:attribute name="id" type="xs:int"></xs:attribute>
                   </xs:complexType>
               </xs:element>
               <xs:element name="instanceDest">
                  <xs:complexType>
                     <xs:attribute name="id" type="xs:int"></xs:attribute>
                  </xs:complexType>
               </xs:element>
               <xs:element name="converters">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="converter" maxOccurs="unbounded">
                           <xs:complexType>
                              <xs:attribute name="serviceId" type="xs:int"></xs:attribute>
                           </xs:complexType>
                        </xs:element>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
               <xs:element name="externalService" type="xs:string"></xs:element>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
   </xs:schema>
```

Apêndice B

Obtenção de informação dos conversores

B.1 XSD de obtenção de informação sobre os conversores

```
<?xml version="1.0" encoding="UTF-8"?>
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
   attributeFormDefault="unqualified">
     <xs:element name="gateway-communication-contents">
        <xs:complexType>
           <xs:sequence>
              <xs:element name="gateway-communication-converter">
                 <xs:complexType>
                    <xs:sequence>
                       <xs:element name="converter" maxOccurs="unbounded">
                          <xs:complexType>
                             <xs:sequence>
                                <xs:element name="name" type="xs:string"></xs:element>
                                <xs:element name="description" type="xs:string"></xs:element>
                                <xs:element name="code" type="xs:string"></xs:element>
                             </xs:sequence>
                             <xs:attribute name="id" type="xs:int"></xs:attribute>
                          </xs:complexType>
                       </xs:element>
                    </xs:sequence>
                 </xs:complexType>
              </xs:element>
              <xs:element name="gateway-communication-instance">
                 <xs:complexType>
                    <xs:sequence>
                       <xs:element name="instance" maxOccurs="unbounded">
                          <xs:complexType>
                             <xs:sequence>
                                <xs:element name="module" type="xs:int"></xs:element>
                                <xs:element name="description" type="xs:string"></xs:element>
                                <xs:element name="src-gateway" type="xs:int"></xs:element>
                                <xs:element name="dest-gateway" type="xs:int"></xs:element>
                                <xs:element name="external-service" type="xs:string"></xs:element>
                             </xs:sequence>
                             <xs:attribute name="id" type="xs:int"></xs:attribute>
                          </xs:complexType>
                       </xs:element>
                    </xs:sequence>
                 </xs:complexType>
              </xs:element>
           </xs:sequence>
        </xs:complexType>
     </xs:element>
   </xs:schema>
```

B.2 XML de resposta do serviço de obtenção de informação sobre os conversores

```
<?xml version="1.0" encoding="utf-8"?>
<gateway-communication-contents>
  <gateway-communication-converter>
    <converter id="12">
      <name>SampleConverter</name>
      <description>Convert integer to string</description>
      <code> (...) </code>
    </converter>
  </gateway-communication-converter>
  <gateway-communication-instance>
    <instance id="22">
      <module>12</module>
      <description>SampleInstance</description>
      <src-gateway>3</src-gateway>
      <dest-gateway>34</dest-gateway>
      <external-service>http://externalService.sample</external-service>
    </instance>
  </gateway-communication-instance>
</gateway-communication-contents>
```

Bibliografia

[1] OSGi Alliance. Osgi. `http://www.osgi.org/Main/HomePage`. Página visitada no dia 22 de Novembro de 2012.

[2] ZigBee Alliance. Zigbee. `http://www.zigbee.org/`. Página visitada no dia 8 de Junho de 2013.

[3] Ardunio. Ardunio. `http://www.arduino.cc/`. Página visitada no dia 28 de Junho de 2013.

[4] Dr. Faisal Aslam. Takatuka. `http://sourceforge.net/apps/mediawiki/takatuka/index.php?title=Main_Page`. Página visitada no dia 22 de Novembro de 2012.

[5] Atmel. Atmel avr-cpu. `http://www.atmel.com/`. Página visitada no dia 28 de Junho de 2013.

[6] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[7] Philip Baldwin, Sanjeev Kohli, Edward A. Lee, Xiaojun Liu, and Yang Zhao. Modeling of sensor nets in Ptolemy II. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, IPSN '04. ACM, 2004.

[8] Niels Brouwers, Koen Langendoen, and Peter Corke. Darjeeling, a feature-rich VM for the resource poor. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 169–182. ACM, 2009.

[9] Choon-Sung Nam and Hee-Jin Jeong and Dong-Ryeol Shin. Design of wireless sensor networks middleware using the publish/subscribe paradigm. In *Proceedings of IEEE International Conference on Service Operations and Logistics, and Informatics (IEEE/SOLI)*, pages 559–563, 2008.

[10] JBoss Community. Jboss hibernate. `http://www.hibernate.org/`. Página visitada no dia 22 de Novembro de 2012.

[11] Moog Crossbow. Crossbow. `http://www.xbow.com/`. Página visitada no dia 8 de Junho de 2013.

[12] Scott de Deugd, Randy Carroll, Kevin Kelly, Bill. Millett, and Jeffrey Ricker. Soda: Service oriented device architecture. *Pervasive Computing, IEEE*, 5(3):94–96, 2006.

[13] Apache Software Foundation. Apache activemq. `http://activemq.apache.org/`. Página visitada no dia 22 de Novembro de 2012.

[14] Apache Software Foundation. Apache cxf. `http://cxf.apache.org/`. Página visitada no dia 22 de Novembro de 2012.

[15] Apache Software Foundation. Apache maven project. `http://maven.apache.org/`. Página visitada no dia 22 de Novembro de 2012.

[16] Apache Software Foundation. Apache servicemix. `http://servicemix.apache.org/`. Página visitada no dia 22 de Novembro de 2012.

[17] Apache Software Foundation. Apache xmlbeans. `http://xmlbeans.apache.org/`. Página visitada no dia 22 de Novembro de 2012.

[18] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[19] Google. Python-on-a-chip. `http://code.google.com/p/python-on-a-chip/`. Página visitada no dia 22 de Novembro de 2012.

[20] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth. The internet of things in an enterprise context. In John Domingue, Dieter Fensel, and Paolo Traverso, editors, *Future Internet Symposium (FIS2008), Vienna, Austria*, volume 5468 of *Lecture Notes in Computer Science*, pages 14–28. Springer, 2008.

[21] Harbaum. Nano VM. `http://www.harbaum.org/till/nanovm/index.shtml`. Página visitada no dia 22 de Novembro de 2012.

[22] Yi Huang and Dennis Gannon. A comparative study of web services-based event notification specifications. In *ICPP Workshops*, pages 7–14. IEEE Computer Society, 2006.

[23] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. SenseWeb: an infrastructure for shared sensing. *IEEE MultiMedia*, 14(4):8–13, 2007.

[24] Oracle Labs. Sunspot. `http://www.sunspotworld.com/`. Página visitada no dia 28 de Junho de 2013.

[25] Philip Levis and David Culler. Maté: a tiny virtual machine for sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(5):85–95, October 2002.

[26] José Alves Marques, Paulo Ferreira, Carlos Nuno da Cruz Ribeiro, Luís Veiga, and Rodrigo Rodrigues. *Sistemas Operativos*. FCA, LIDEL, March 2009.

[27] Francisco Martins, Luís Lopes, and João Barros. Towards the safe programming of wireless sensor networks. In *Proceedings of PLACES'09 - Programming Language Approaches to Concurrency and Communication-cEntric Software*, volume 17 of *EPTCS*, pages 49–62, 2010.

[28] MySQL. Mysql. `http://www.mysql.com/`. Página visitada no dia 22 de Novembro de 2012.

[29] The Open Geospatial Consortium (OGC). Sensor model language. `http://www.opengeospatial.org/standards/sensorml`. Página visitada no dia 8 de Junho de 2013.

[30] The Open Geospatial Consortium (OGC). Sensor web enablement. `http://www.opengeospatial.org/ogc/markets-technologies/swe`. Página visitada no dia 8 de Junho de 2013.

[31] Hervé Paulino and João Ruivo Santos. A middleware framework for the web integration of sensor networks. In *Proceedings of S-Cube 2010 - The 2nd International ICST conference on Sensor Systems and Software*, pages 75–90, 2010.

[32] Doug Simon and Cristina Cifuentes. The squawk virtual machine: Java on the bare metal. In Ralph E. Johnson and Richard P. Gabriel, editors, *Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, October 16-20, 2005, San Diego, CA, USA*, pages 150–151. ACM, 2005.

[33] Fuse Source. Fuse ESB. `http://fusesource.com/products/enterprise-servicemix/`. Página visitada no dia 22 de Novembro de 2012.

[34] Bruno Valente and Francisco Martins. A middleware framework for the internet of things. In *AFIN 2011*, volume 57 of *The Third International Conference on Advances in Future Internet*, page 139 to 144, 2011.

[35] Bruno Alexandre Loureiro Valente. Um middleware para a Internet das coisas. Master's thesis, Universidade de Lisboa, Faculdade de Ciências.

[36] Mohammad Hadi Valipour, Bavar Amirzafari, Khashayar Niki Maleki, and Negin Daneshpour. A brief survey of software architecture concepts and service oriented

architecture. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference*, pages 34–38, 2009.

[37] Duarte Vieira and Francisco Martins. Integrating WSN simulation into workflow testing and execution. In *Proceedings of S-Cube 2010 - The 2nd International ICST conference on Sensor Systems and Software*, LNICST. Springer, 2010.

[38] Deze Zeng, Song Guo, and Zixue Cheng. The web of things: A survey (invited paper). *JCM*, 6(6):424–438, 2011.